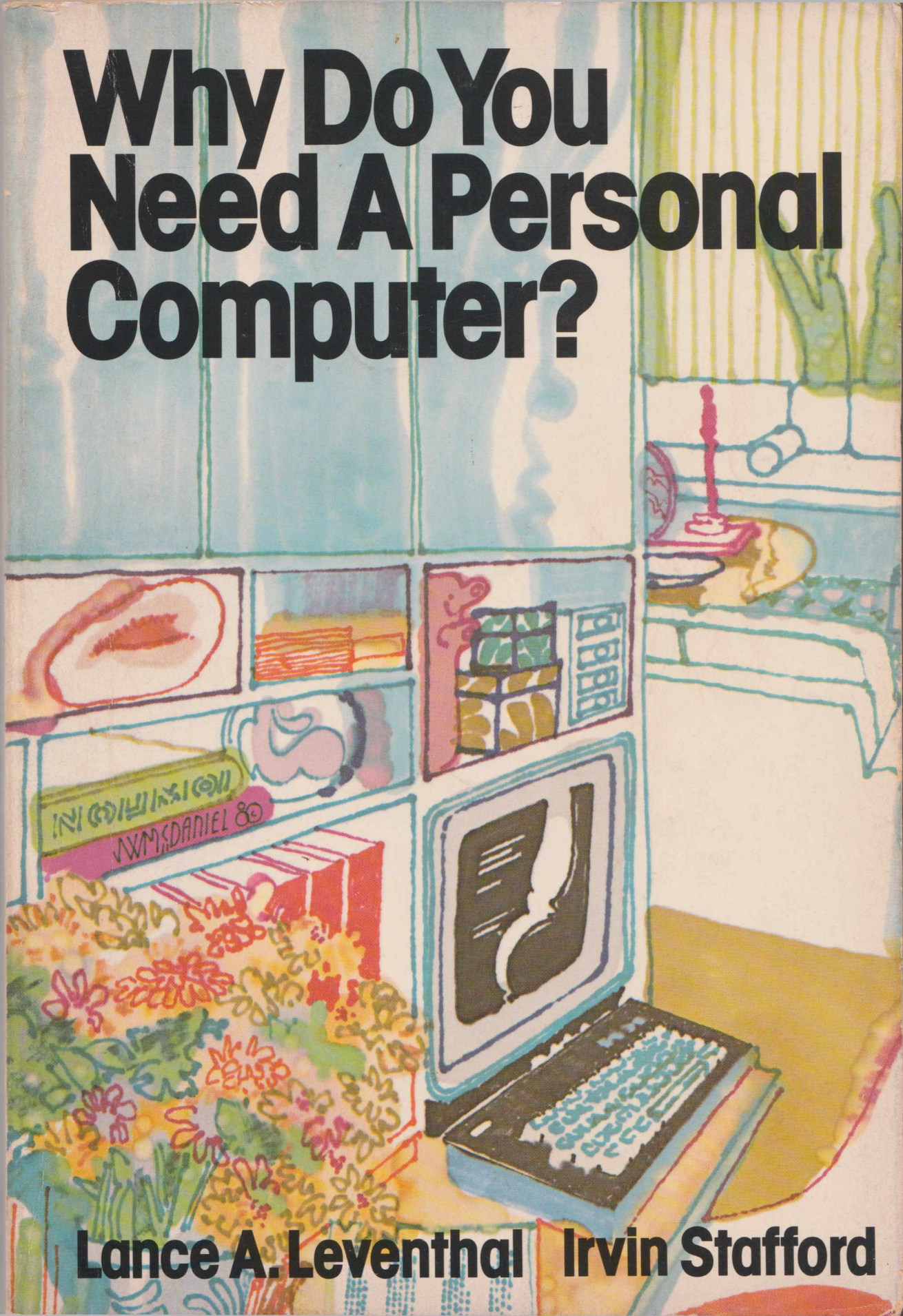
Why Do You Need A Personal Computer?



Lance A. Leventhal Irvin Stafford

Leventhal Stafford

WHY DO YOU NEED A PERSONAL COMPUTER?

Wiley

G. DOSA

WHY DO YOU NEED A PERSONAL COMPUTER?

LANCE A. LEVENTHAL
Consultant, Emulative Systems Company, Inc.,
San Diego, CA

IRVIN STAFFORD
Senior Engineer, Burroughs Corporation
Advanced Systems Development Organization,
San Diego, CA

John Wiley & Sons, Inc.
New York • Chichester • Brisbane • Toronto

Copyright © 1981, by John Wiley & Sons, Inc.

All rights reserved. Published simultaneously in Canada.

Reproduction or translation of any part of this work beyond that permitted by Sections 107 or 108 of the 1976 United States Copyright Act without the permission of the copyright owner is unlawful. Requests for permission or further information should be addressed to the Permissions Department, John Wiley & Sons, Inc.

Library of Congress Cataloging in Publication Data

Leventhal, Lance A 1945-
Why do you need a personal computer?

Includes index.
1. Microcomputers. 2. Minicomputers. I. Stafford, Irvin, joint author. II. Title.
QA76.5.L489 001.64 80-23091
ISBN 0-471-04784-8

Printed in the United States of America

81 82 10 9 8 7 6 5 4 3 2

Preface

The personal or home computer is a reality. Since their introduction in 1975, personal computers have grown into a multimillion dollar industry, with leading manufacturers reporting sales of hundreds of thousands of units. Computers are now commonly advertised on radio and television and in newspapers and magazines.

Still, most people find computers mystifying. Many potential users are afraid of them, believing them to be complicated, expensive, and even domineering. They have not accepted the computer as a useful tool, much like the typewriter or the pocket calculator, but capable of doing far more varied tasks and acting in an intelligent manner. The world of computers is not only closed to most people, but appears to them to be completely cut off from their own problems and interests.

This book discusses personal computers from the point of view of the potential user. We emphasize insights into what terms mean and why they are important, what is needed for actual applications, and what are the current strength and weaknesses of computer technology. Our aim is to make the reader an intelligent consumer, not an expert on computers. We have provided guidelines on how to write programs, how to select computers and peripherals, how to maintain computers, and how to determine the kind of computer best suited to a specific application. We want the reader to understand why personal computers are so popular, as well as what problems are yet to be solved. Such a reader will be able to evaluate current and future developments and to take advantage of advancing computer technology.

An understanding of computers will be of ever-increasing importance in fields as diverse as business, government, education, medicine, and engineering. Soon a fear of computers will seem as ridiculous as the earlier fears of automobiles, airplanes, or electricity. Computers will become commonplace, and those who become familiar with them early will have a tremendous advantage.

Our book is organized as follows:

Chapter 1 describes how ordinary people can use personal computers, how the cost of computers has been reduced, how a computer operates, and how computers do "intelligent" things, such as answering questions and greeting people by name. Chapter 1 also compares different types of computers and shows the differences between personal computers and larger machines.

Chapter 2 discusses the requirements for a useful computer. It describes the types of personal computers and their advantages and disadvantages, and shows how the various requirements are met in personal computers.

Chapter 3 introduces the BASIC computer language. It builds on three examples (an arithmetic quiz, inventory control, and a board game), showing how to make a computer perform simple calculations, report the results clearly, make decisions, repeat actions, handle large amounts of data, interact with an operator, analyze data, and handle nonnumerical concepts.

Chapter 4 describes the problems involved in writing programs. This chapter discusses how to define a program's functions, design the program, discover and correct errors, test the program, and document it. The chapter also describes modern programming methods and tools that make program writing more efficient.

Chapters 5 and 6 describe peripherals and interfacing, respectively. Chapter 5 describes the peripherals required for a useful computer, their typical features and options, their advantages and disadvantages, and the selection of peripherals to meet particular needs. Chapter 6 describes interfacing or the connecting of peripherals to a computer. The topics discussed include the hardware and software interfaces, standard interfaces, selection of interfaces, and the interfaces commonly available for popular peripherals and computers.

Chapter 7 deals with the operation and maintenance of computers and peripherals. It discusses the required environment, the installation and operation of computers, and the care and maintenance of the computer, its peripherals, and its software. Chapter 7 also describes how to recognize and isolate problems, what tools and background you need to fix a computer, and what to do if you cannot fix the computer yourself.

Chapter 8 discusses the selection of a computer. It describes where to find equipment and information, how to select a computer for various applications, and what questions to ask in evaluating computers. It includes a few simple case studies about purchasing a computer.

The Appendixes present background material, including code tables, descriptions of electrical components, interface standards, magnetic recording methods, television standards, and flowchart symbols, Appendix 1 is an extensive glossary.

We have emphasized discussions of the reasons behind products, trends, and problems in the personal computer industry. Many books describe specific computers and accessories without explaining why certain products are particularly popular, why users prefer certain methods, and what problems have not yet been solved. Specific manufacturers and products change rapidly, but basic problems such as the difficulty of producing or obtaining software, the lack of universal standards, and the need for more storage and useful, well-written programs will remain.

We do not, however, want to overrate the problems of computers. The personal computer is a marvelous, exciting achievement. Today for a few hundred dollars a person can buy a machine that is more powerful, easier to use, more reliable and has more accessories than a large computer of the 1950s that cost hundreds of thousands of dollars. Surely no other industry can claim comparable progress. The fundamental question that remains to be answered is how we can use personal computers to improve the quality of life.

Acknowledgments

Many people helped and encouraged us in the writing of this book. We would like particularly to thank our wives, Donna Leventhal and Rosemary Stafford, for their patience and understanding. We would also like to thank our editors: Judy Green, Dianne Littwin, and the originator of this project, Irv Kosow. Others who deserve thanks include Gary Hankins, Winthrop Saville, Romeo Favreau, and Marielle Carter of Sorrento Valley Associates; Michael Lehman of MT Microsystems; Norm Dinnsen, Dave McDonough, Greg Grumich, and Ed Faber of Computerland; David Hata of Portland (OR) Community College; Sheila Clarke of InfoWorld; Allen Katz of Trenton State College; Sol Libes of Union County Technical Institute; Arthur Seidman of Pratt Institute; and Louis Covert of Southern Technical Institute (GA). Marielle Carter and Jacqueline Roberge did much of the typing. Many manufacturers and suppliers provided us with information and pictures; we would like particularly to thank Alpha Micro, Apple Computer, Atari, Centronics, Compucolor, Cromemco, Durango Systems, Exidy, Global Specialties, Heath, Houston Instrument, IMS Associates, Intel, Intelligent Systems, Motorola, MOS Technology, North Star, Ohio Scientific Instruments, Pertec, Pickles and Trout, Radio Shack, Southwest Technical Products, Teletype, Texas Instruments, Thinker Toys, and Vector Graphic. An anonymous reviewer criticized a nearly completed version of this book with great care and insight, forcing us to revise a major part of it; that reviewer deserves a special acknowledgment.

San Diego, CA

Contents

1 YOUR OWN COMPUTER 1
How Can You Use a Computer?, 6
A Little History, 11
Parts of a Computer, 13
Hardware and Software, 16
Computers and Mathematics, 18
How Does the Computer Work?, 19
How Can Computers Seem To Be so Smart?, 20
How Do Computers Differ?, 25
References, 30

2 COMPONENTS OF A COMPUTER 31
Hardware, 32
Software, 35
Data Bases, 37
Compatibility, 37
Categories of Personal Computers, 38
One-Board Computers, 38
Appliance Computers, 42
Modular Computers, 45
Central Processing Unit, 50
RAM, 52
ROM, 53
Operating Systems, 53
Assemblers, 54
Compilers and Interpreters, 55
Editors, 56
Debuggers, 57
Summary, 57

3 INTRODUCTION TO BASIC PROGRAMMING 59
Problem Examples, 59
Introduction To BASIC, 60
Initial Programs, 61
Brief Review 1, 64
Repeating Operations, 65
Brief Review 2, 68
Counting, 68
Making Decisions, 70
Brief Review 3, 76
Interaction, 77
Assigning Numerical Values, 81
Loops, 84
Brief Review 4, 89

Processing Sets of Data, 89
Random Numbers, 97
Brief Review 5, 98
Conclusion, 98
References, 99

4 HOW TO WRITE PROGRAMS 100
The Stages of Program Writing, 101
Problem Definition—The Starting Point, 102
Program Design, 104
A Design Procedure, 108
Top-Down Design, 108
Modular Programming, 116
Structured Programming, 117
Coding, 120
Debugging, 122
Testing—Does the Program Really Work?, 130
Documentation, 131
Summary, 134
References, 135

5 PERIPHERALS 136
Standard Peripherals, 137
The Keyboard—A Simple Input Device, 137
The Video Display—A Simple Output Device, 141
Mass Storage Devices, 144
Cassette Recorders, 145
Floppy Disks, 146
Hard Disks, 150
Choosing a Disk, 151
Printers, 152
Factors in Selecting a Printer, 153
General Characteristics of Printers, 157
Printers for Various Applications, 160
Other Peripherals, 161
Summary, 164

6 INTERFACING 165
The Hardware Interface, 166
Comparing Hardware Interfaces, 167
Building Interfaces, 168
Standard Hardware Interfaces, 170
The Software Interface, 171
Hardware/Software Tradeoffs, 173
Input/Output Boards, 173
Choosing an Interface, 174
Interfaces for Various Peripherals, 174
Interfaces for One-Board Computers, 178

Interfaces for Appliance Computer, 180
Interfaces for Modular Computers, 181
Summary, 182

7 COMPUTER OPERATION AND MAINTENANCE 183
The Operating Environment, 185
Setting up the Computer, 186
Initial Operation, 188
Computer Maintenance, 190
Cassette Maintenance, 190
Floppy Disk Maintenance, 192
Keyboard Maintenance, 193
CRT Maintenance, 193
Printer Maintenance, 194
Recognizing Problems, 194
Isolating the Problem, 195
Correcting the Problem—Should You Try to Fix It?, 196
Service Contracts, 196
Restoring Operations, 197
What Should You Know to Attempt Repair?, 197
The Repair Environment, 197
Software Maintenance, 200
Summary, 202

8 SELECTING A COMPUTER 203
The Key Questions, 203
Buying for Your Application, 203
Estimating Costs, 207
Buying for Your Background, 208
Deciding on Your Own Contribution, 209
Planning for the Long Term, 211
Other Questions, 216
Sources of Information, 217
Computer Magazines, 217
Computer Clubs, 220
Computer Stores, 220
Computer Shows, 221
Schools, 222
Mail Order Suppliers, 222
Other Sources, 223
Fitting the Computer to the Task, 223
Summary, 230

Appendix 1 GLOSSARY 231

Appendix 2 CODES 247

Appendix 3 SIMPLE ELECTRICAL COMPONENTS 249

Appendix 4 INTERFACE STANDARDS 252
RS-232, 252
IEEE-488, 253
Centronics Interface, 254
S-100 Bus, 256
Heath H8 Bus, 256
Radio Shack Bus, 256
SS-50 Bus, 256
Apple I/O Bus, 256
KIM Bus, 256
OSI-48 Bus, 256

Appendix 5 GENERAL MAGNETIC RECORDING TECHNIQUES 265
Cassette Recording Techniques, 265
Types of Flexible (Floppy) Disks, 266

Appendix 6 TELEVISION SIGNAL STANDARDS 267

Appendix 7 STANDARD FLOWCHART SYMBOLS 268

INDEX 269

1 Your Own Computer

Hey, buddy, want to buy a computer? Just a few years ago, that idea was as outlandish as buying your own aircraft carrier or gold mine. Not only were computers incredibly expensive (even by modern inflated standards) but they were also enormous and complicated. Only a mathematical genius could really make one work. As far as most of us were concerned, all a computer did was reduce us to a few holes on a punched card, marked "Do not fold, tear, bend, or mutilate."

Today, however, you can go into a computer store, radio/electronics store, or department store and buy your own computer. What does the computer look like? Without any accessories, it probably looks like the Cromemco Z-2 in Figure 1-1, about the size of a stereo receiver or a large tape recorder. Some personal computers, like the Commodore PET in Figure 1-2, include a television set and a cassette recorder, all packaged in a single unit like a complete stereophonic system. Other computers, like the Southwest Technical Products 6800 (Figure 1-3) or Radio Shack TRS-80 (Figure 1-4), resemble component stereo systems with the television set, cassette recorder, and other accessories packaged separately. Of course, most stores offer additions like those shown in Figures 1-3, 1-5, 1-6, and 1-7.

How much does a computer cost? You can buy the one in Figure 1-2 for under $1,000; the more elaborate model in Figure 1-8 costs several thousand dollars. As usual, there are extras (and essentials) that raise the actual cost. But a computer still is comparable in price to a home entertainment center or a camping outfit. An average person really can afford to buy a computer.

Of course, the obvious questions are

—What would I do with a computer?
—How do I tell the computer what I want it to do?
—What computer and what accessories should I buy?

The rest of this book answers those three questions. The present chapter describes how personal computers can be used, how they work, and how computers differ from one another. Later chapters describe what types of computers and accessories are available, how to tell the computer what to do, what you can attach to the computer, as well as how and what to look for when you buy a computer.

Figure 1-1. The Cromemco Z-2 computer (Courtesy of Cromemco Inc., Mountain View, CA)

Figure 1-2. The Commodore PET personal computer with integral video display (television screen), keyboard, and cassette recorder (Courtesy of Commodore Business Machines, Santa Clara, CA)

Figure 1-3. The Southwest Technical Products SWTP 6800 computer system. The peripherals are stacked on top of the computer itself (Courtesy of Southwest Technical Products Corp., San Antonio, TX)

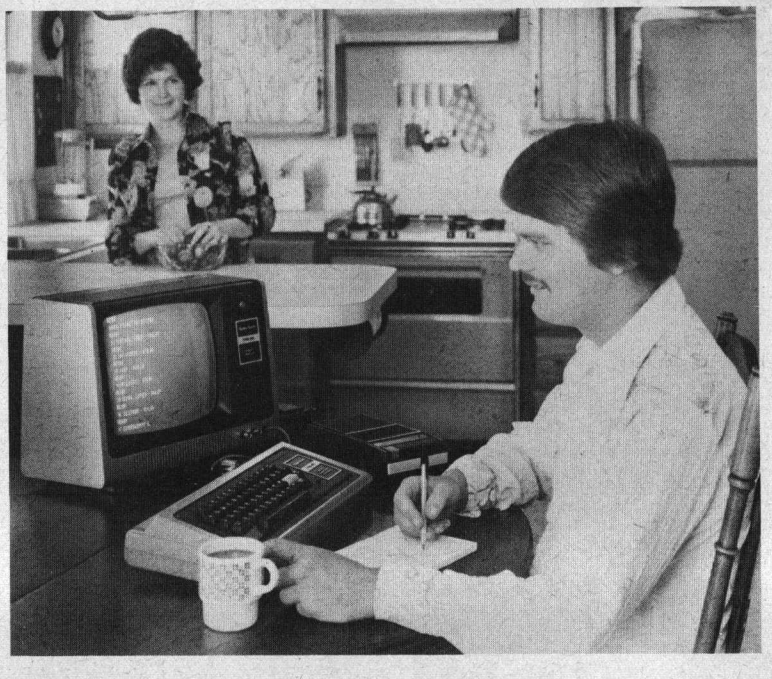


Figure 1-4. The Radio Shack TRS-80 microcomputer (Courtesy of Radio Shack Division, Tandy Corp., Fort Worth, TX)

Figure 1-5. The complete line of Atari personal computer products (Courtesy of Atari, Inc., Sunnyvale, CA)

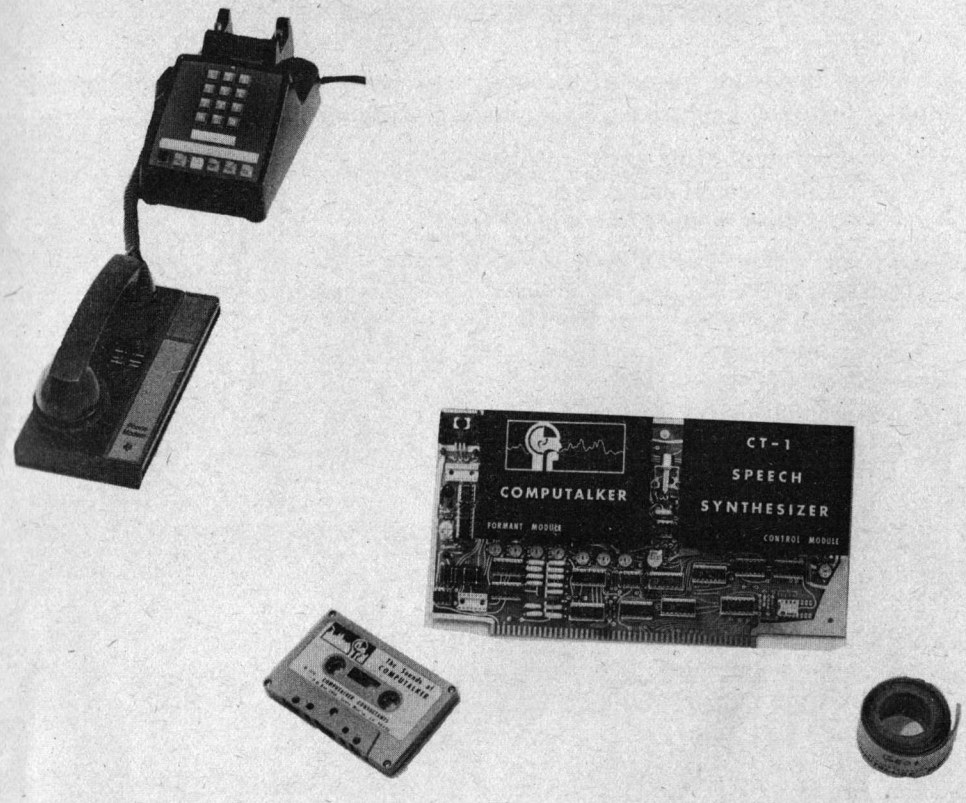


Figure 1-6. The PHP1600 Telephone Coupler (Modem) allows the Texas Instruments 99/4 Home Computer to communicate over telephone lines. (Courtesy of Texas Instruments, Inc., Dallas, TX)

Figure 1-7. The Computalker Consultants CT-1 Speech Synthesizer. This device allows a computer to produce natural-sounding speech. (Courtesy of Computalker Consultants, Santa Monica, CA)

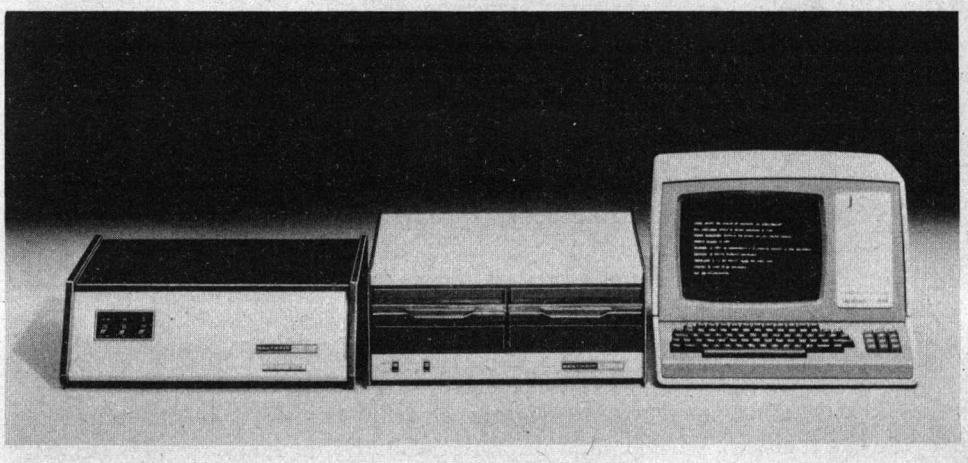


Figure 1-8. A Heath computer system with the H11 computer, H27 dual floppy disks, and H19 intelligent terminal (Courtesy of Heath Co., Benton Harbor, MI)

HOW CAN YOU USE A COMPUTER?

Let's begin by listing some of the uses that have appeared in two popular computer hobbyist magazines, *Byte* and *Microcomputing*.

—School accounting and recordkeeping
—Calculating golf handicaps
—Controlling a ham radio station
—Apartment management
—Chess game
—Figuring costs for construction jobs
—Mailing lists
—Electronic mail
—Air conditioning controller
—Inventory control
—Lunar lander game
—Adventure game
—Three-dimensional graphics
—Automated design of electrical circuits
—Pharmacy management
—Structural analysis for civil engineers
—Baseball and football games
—Biorhythm generator
—Star Trek game
—Classroom drill and practice

Games

Many people use personal computers to play games. By now you've probably played an electronic game in a club, restaurant, airport, or on your own television set. The games are fun, but the variety is limited. You know the old story: first time—boy, that's great; second time—terrific, can't wait to get back to it; tenth time—that old thing again. Even elaborate games like the Bally Alley game shown in Figure 1-9 become dull after a while.

With your own computer, however, you can produce your own games.

—You can choose the rules and change them when you want. (That keeps the other players alert).
—You can choose your own game: auto racing, word games, mazes, horseshoes, stock market, home economics, ecology, or energy.
—You can try games from books or magazines, simply by reading them into your computer.
—You can have the game talk to you or listen to you; you can save the results on a cassette and find out where you went wrong; you can practice by yourself and learn how to beat everybody (or lose to everybody if you're a masochist).

The key to using the computer for games is its flexibility. A computer can provide the realisim, variety, and unpredictability that makes games exciting and challenging. The computer can give you a home entertainment center that's just as fascinating and as diversified as the ones that science fiction writers describe.

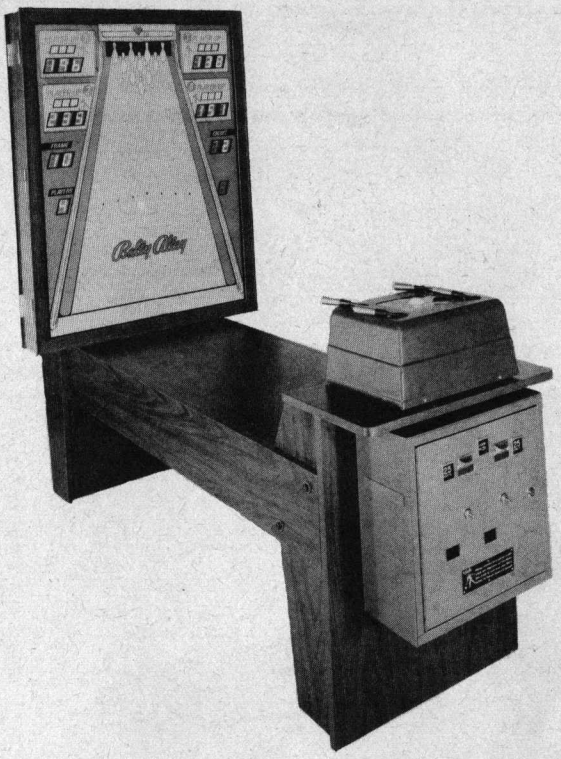


Figure 1-9. The Bally Alley bowling game, a computer-controlled pinball machine (Courtesy of Bally Manufacturing Co., Chicago, IL)

Education

Computer-aided instruction is not a new idea, but the high cost of computers has limited its use. With your own computer, however, you can learn at your own pace without anybody looking over your shoulder. Do you want to

—Practice those pesky fractions and decimals?
—Study for a driving test?
—Learn a foreign language for a trip?
—Study for a licensing examination?
—Study law or retirement plans?
—Learn the building code or zoning rules?
—Learn to play chess or bridge or poker?
—Learn a craft or skill?
—Learn about the stock market, options, commodities, or real estate?

A computer can help you with any of these tasks. It can display the questions, tell you whether your answer was correct, offer you hints, and grade you (or not grade you, if you prefer). The computer never becomes bored or impatient. It is willing to continue as long as you want and to repeat the same material as many times as you can endure it.

Furthermore, you can teach as well as learn. Want to help someone learn to spell, divide, operate a camera, or build a model airplane? Would you like an easy way to teach newcomers how to operate a cash register, handle credit

cards, or fill out reports? Just put the information in the computer; it will be there whenever you need it or the student is ready for it.

If you're a teacher by profession, you may want to use your computer to generate and grade tests, provide remedial or extra credit lessons, and keep your own skills sharp. Even if you're not a teacher, you may still want to use the computer for training new employees or volunteers, helping those who are handicapped or do not speak English, or working with youth or church groups. The computer does not replace the teacher, but it allows the teacher to escape from mundane work and concentrate on tasks that require personal attention.

Business and Finance

If you own a small business, you can surely think of things that you would like a computer to do, such as:

Accounts payable
Accounts receivable
Billing
General ledger
Insurance forms
Inventory
Mailing lists
Payroll
Permit forms
Progress reports
Tax records

A small business has the same problems as a larger business, including the problem of adequate help. A computer is help that doesn't get paid, switch jobs, go on vacation, stay home sick, or become your competitor. A personal computer can do anything that a big computer can do, only on a smaller scale. If you own a small business, or are an independent professional or contractor, that smaller scale may be just what you need.

Even if you're not in business, you've got financial problems to solve. What are the alternative ways you can finance a new house, a new car, or a vacation? How are your investments doing and how do the new possibilities stack up? How much must you save per month to accumulate a down payment? What's the current state of your bills, your mortgage, your checking and savings accounts? What's the best way to reduce your taxes, plan for retirement, buy insurance, or handle your estate? Big companies don't guess at things and keep records on scraps of paper; they use computers. So can you. The computer can organize your records and produce information when you need it.

Word Processing

Wouldn't you like to have a typewriter that lets you easily correct spelling and punctuation, insert or delete an extra word or line, and adjust the margins and spacing before it prints the final version? What if it could store form letters so you only had to press a few keys to print one? Figure 1-10 shows

Figure 1-10. The Redactron Redactor II editing typewriter, a word processing system (Courtesy of Redactron Corp., Hauppauge, NY)

the computerized typewriter (or *word processing system*) that can do all those things.

If you're in business, a word processing system can produce professional correspondence, accurate invoices, and your own advertising letters. If you're a teacher, it can produce lesson plans or fill our forms. If you're a student, it can make your homework legible. You can produce a resume, a calendar, a list of instructions, a magazine article, a book, a brochure, or an advertising flyer. You can see what you have typed on the television screen and make corrections before you commit anything to paper. Furthermore, you can save your work and use it again later.

Art and Music

Of course, a computer is not restricted to numbers and words; it is equally at home with musical notes, sketches, building plans, needlepoint patterns, or model airplane designs. Try the plans or patterns on the computer first and see how they look or sound. Then save them on a cassette and see if you still like them later or if other people approve of them.

The computer can particularly help an art or music teacher, since it can produce the patterns or sounds that are so difficult to describe in words. A computer can help plan an advertising campaign or package design; imagine being able to see your thoughts in color on a display like the one in Figure 1-11. The computer can be as important a tool for the artist or designer as it is for the businessman or engineer.

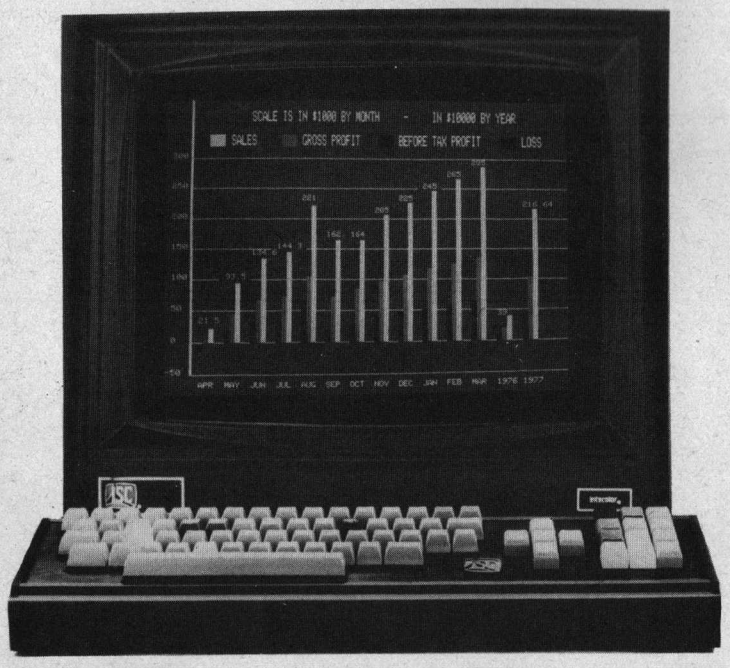

SCALE IS IN $1000 BY MONTH - IN $10000 BY YEAR
SALES
GROSS PROFIT
BEFORE TAX PROFIT
LOSS
APR MAY JUN JUL AUG SEP OCT NOV DEC JAN FEB MAR 1976 1977

Figure 1-11. The Intecolor 8051 color terminal (Courtesy of Intelligent Systems Corp., Norcross, GA)

Hobby Helper

Maybe you've already got a hobby—golf, tennis, bridge, stamp collecting, amateur radio, or model trains. The computer can help you control your equipment, improve your skills, and keep records. Want to know what your average or handicap is, or how much you'll have to improve to reach that next level? Want to learn how to be a league director, tournament director, or official? Want to know if you and your partner have that new bridge system down pat? Want to know what stamps you've already got and what they're worth in the latest catalog? Want to have your radio station automatically record your messages? A computer can do all this and more.

The computer can also

—Keep track of your progress.
—Show you the effects of a move, decision, or strategy.
—Produce player rankings and league records.
—Handle scheduling and scoring.

Simulation

Another major use of large computers is simulation. This is modeling real things to predict their future behavior and to understand how they will be affected by new designs, new policies, and outside events. Simulators are widely used to train airplane pilots, astronauts, and reactor technicians for whom real experience is expensive and difficult to obtain. Simulations are also used to test preparations for emergencies and disasters.

You, too, can use computer simulation. Games are a form of simulation, but you may want to use more serious forms like those that large corporations employ in their planning departments. Would you like to estimate next year's taxes or next year's sales for your business? How will a new traffic light, a new freeway, or a new shopping center affect traffic congestion in your neighborhood or the number of customers in your store? How much will a zoning change or a remodeling project affect your property taxes? What is the risk involved in starting a new business, entering into a new investment, or introducing a new product? How should changes in interest rates, the rate of inflation, or the state of the economy affect your financial plans? Computer simulation can predict the likely results of your plans and decisions; it can also allow you to try different strategies and see how they turn out.

Other Uses

We'll just list a few because we're sure you'll be able to think of many more.

—Driver training
—Menu or diet planning
—Home or business security system
—Travel planning
—Telephone answering system
—Appointment reminder
—Stock market or commodities analysis
—Stress calculations for building design
—Medical, dental, and legal accounting and office management
—Files for clubs, associations, churches, and youth groups
—Automatic reminders for service calls, license renewals, and scheduled maintenance
—Computer art or portraits
—Water and energy monitors
—Interviewing and screening patients or clients
—Insurance rating
—Income tax preparation
—Capital equipment budgeting
—Surveying and civil engineering

The key to a computer, as we shall see again and again, is that it can do all these things. The only limit is your imagination. Now that computers are readily available, surely we will see applications far more incredible than anything science fiction writers have proposed.

A LITTLE HISTORY

An obvious question is why computers have become less expensive while everything else has gone up on price. The answer lies in the invention of what is called an integrated circuit. The original computers contained many separate electrical devices. Such computers were expensive because building them required a large number of components and a large amount of labor; they also occupied a lot of space, used a lot of power, produced a lot of heat, and required air conditioning and special wiring. Furthermore, the large number of

devices and connections made the computer unreliable, much like old television sets and household appliances.

Integrated circuits now combine many electrical devices into a single small package called a *chip*. Currently, in fact, semiconductor manufacturers can place many thousands of electrical devices on a single chip less than a quarter of an inch long. Figure 1-12 shows a wafer containing many chips, each of which can act as the entire brain of a computer. The electrical devices involved would have cost thousands of dollars twenty years ago, but today these chips cost only a few dollars apiece.

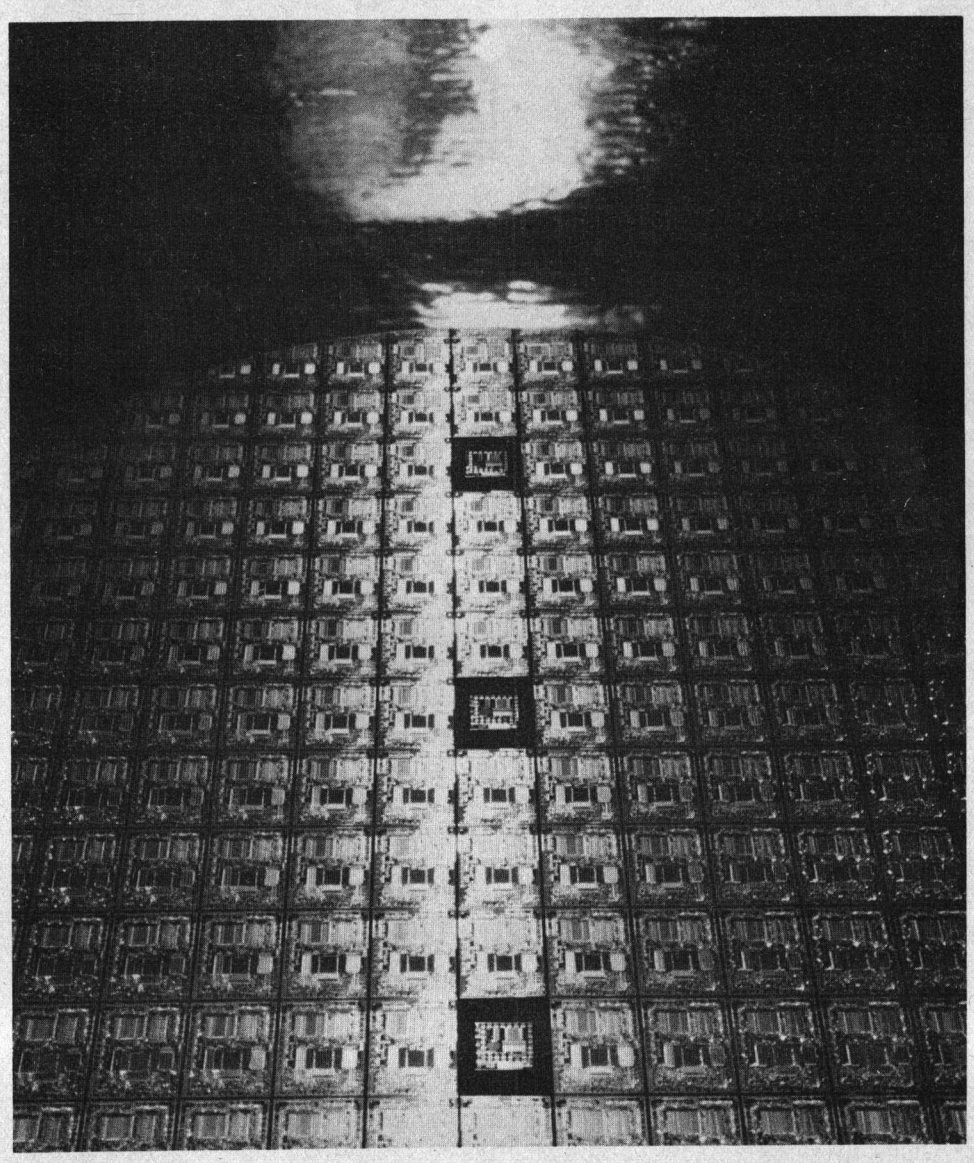


Figure 1-12. A wafer containing Intel 8085 microprocessors (Courtesy of Intel Corp., Santa Clara, CA)

The technique of placing many electrical devices on a chip is called *large-scale integration* or LSI. Manufacturing LSI chips is like developing photographs. The costs are independent of the number of devices on the chips, much as the cost of developing a photograph is independent of the number of things in the picture. A photograph of a crowd scene costs no more to develop than one of an empty sky. So the more devices the manufacturer place on a chip, the lower the total cost of the final product. Fewer separate devices also mean fewer connections, less labor, lower energy consumption, less heat, fewer parts to fail, and simpler maintenance. LSI has produced a $10 calculator that can do more and is easier to use than the $1,000 mechanical calculators of the 1960s. LSI has also given us inexpensive digital watches, clocks, and microwave ovens.

When LSI is used to place the entire brain of a computer on one chip, the resulting device is called a *microprocessor.* A microprocessor is not a computer by itself; alone it is like a person who can think but cannot remember and has no senses or muscles. We must attach memory and input/output circuits to a microprocessor to form a working system or *microcomputer.* Such a computer can do anything that a large computer can do (although perhaps slower). Microcomputers not only serve as personal computers but also have widespread uses in instruments, test equipment, communications, office equipment, military gear, automobiles, appliances, and games.

Most microprocessors are produced by companies that are new and little-known to the general public. The leading manufacturer is Intel, a company founded in the late 1960s. Other unfamiliar names are Mostek, Advanced Micro Devices, American Microsystems, Zilog, and MOS Technology. More familiar names (because they also make consumer goods) are National Semiconductor, Fairchild, Motorola, Texas Instruments, Rockwell, and RCA. The microprocessors most widely used in personal computers are the Intel 8080, Motorola 6800, MOS Technology 6502, and Zilog Z-80.

PARTS OF A COMPUTER

Let us backtrack a little now and see what a computer really is. Computers consist of three parts, as shown in Figure 1-13:

- —*Central processing unit (CPU),* the brain that directs all the computer's activities.
- —*Memory,* the storage place for the data the computer must remember and the programs that tell the computer what to do. Both data and programs are in the computer's memory, much as a person's memory contains both facts and ways to do things.
- —*Input/output (I/O) section,* the connections between the computer and the outside world. The input/output section corresponds to the body's nerves and muscles.

Whereas the CPU is a single unit, the memory consists of many storage places and the input/output section consists of many connections. The basic unit of storage in the memory is called a *word;* we identify a particular word

by its *address* (e.g., word 273 or word 3511). Figure 1-14 shows a memory divided into words. The basic I/O connection (see Figure 1-15) is called a *port;* each port has a number (e.g., input port 3 or output port 5). The internal connections that are used to move things from one part of the computer to another (see Figure 1-13) are called *buses.*

The computer needs not only connections to the outside world, but also actual devices that can accept data and record results; these devices are called *peripherals.* Typical input peripherals are card readers, paper tape

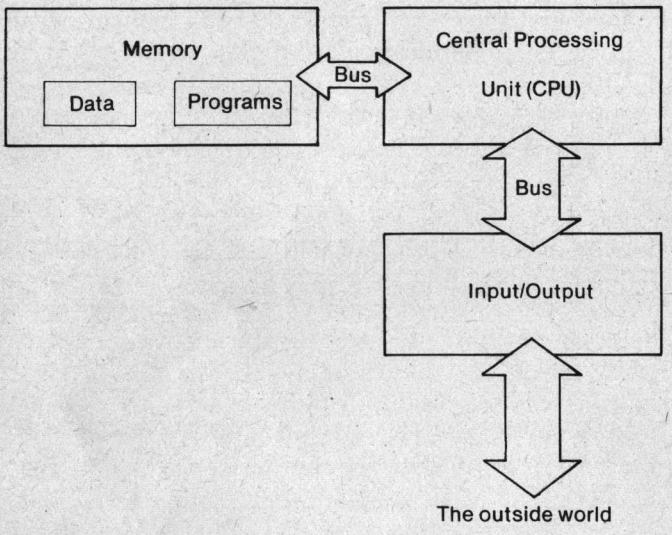


Figure 1-13. Parts of a computer

Address	Contents
•	•
•	•
•	•
•	•
•	•
•	•
•	•
357	42
358	66
359	511
360	0
361	25
•	•
•	•
•	•
•	•
•	•
•	•
•	•

Figure 1-14. The organization of a computer's memory. The address allows the CPU to select a particular word of memory.

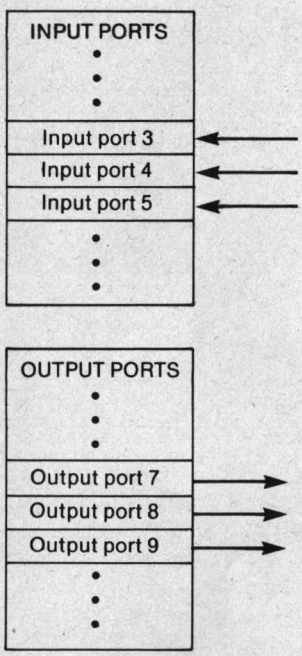


Figure 1-15. The organization of a computer's input/output (I/O) section. The port number allows the CPU to select a particular input or output connection.

readers, and keyboards; typical output peripherals are lighted displays, video displays (like television sets), and printers. Of course, peripherals may handle both input and output; typical examples are cassette recorders and typewriters. Input and output peripherals nowadays often cost more than the computer does, since they contain expensive mechanical parts rather than chips. You should be aware that most advertised computers include few peripherals; you usually have to buy them separately at additional cost.

Many computer peripherals, particularly those used with personal computers, are commonplace devices, such as typewriters, television sets, keyboards, and cassette recorders. A special term is the word *terminal,* used in the computer industry to refer to the combination of an input device and an output device, most commonly a keyboard and a printer (like a typewriter) or a keyboard and a video display (called a video terminal). A useful peripheral that is not commonplace is the *magnetic disk* (see Figure 1-16), a circular object with a magnetic surface. A disk is like a phonograph record except that information is stored on the magnetic surface rather than in grooves. When the computer wants to record information on a disk or read information from it, the computer moves either the head or the disk to the proper position, much as one can move either the needle or the record on a phonograph. So, unlike a tape that may have to be played or rewound, any position on the disk can be reached in about the same amount of time. A small, inexpensive type of disk is the "floppy" disk (see Figure 1-17). Floppy disks are the size of 45-rpm records but are more flexible and can be bent like a piece of thin plastic.

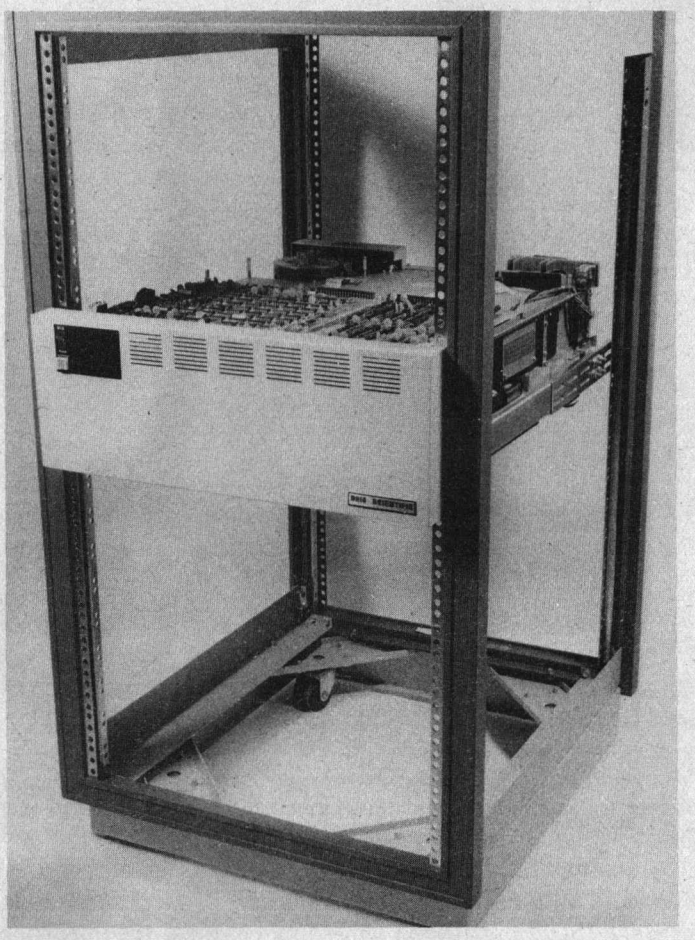


Figure 1-16. A magnetic disk system that stores almost 600 million bits of information (Courtesy of Ohio Scientific Instruments, Aurora, OH)

HARDWARE AND SOFTWARE

Even the peripherals aren't the real key to a computer. All the equipment that we have described (called *hardware*) forms only the skeleton of a computer. What fills in the skeleton and actually allows the computer to perform useful work are the programs or *software* that tell the computer what to do. A computer without software is like a musical instrument without music.

Although constructing computers (like building instruments) is difficult and challenging, writing programs (like composing music) is even more challenging. The programs often are the most expensive part of the computer, much as a record or tape collection is often the most expensive part of a home entertainment system. Furthermore, good programs are as uncommon and as difficult to find as are good records or tapes. Of course you can write your own programs, just as you can produce your own tapes, but that can satisfy only a small part of your needs.

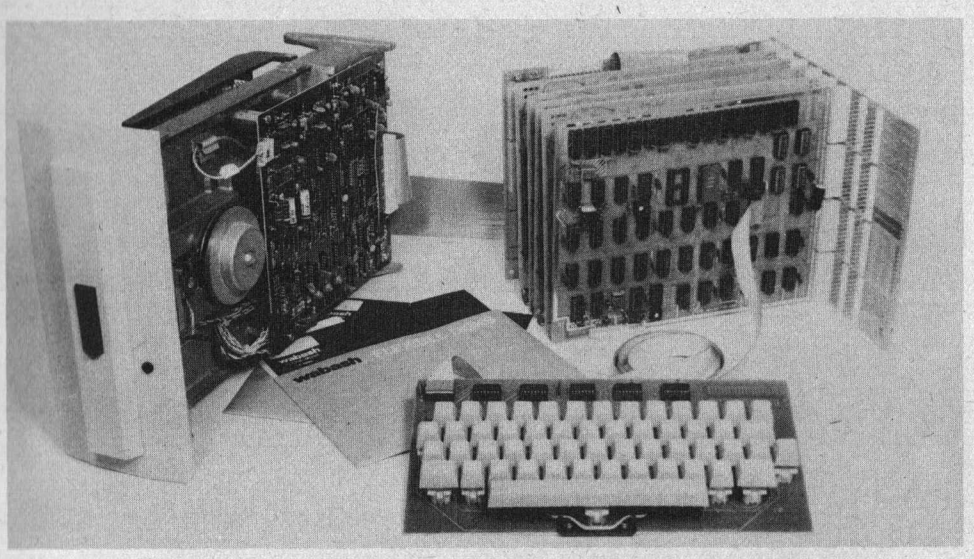


Figure 1-17. The OSI Challenger II C2-8K floppy-disk-based computer system. Two individual floppy or flexible disks (also called diskettes) are in the middle of the picture. (Courtesy of Ohio Scientific Instruments, Aurora, OH)

What is this software and what does it do? Remember that the computer itself consists of electrical circuits like those in a television set, a refrigerator, or a calculator. A computer does not plan its course of action. It merely does what it is told, much as a calculator performs addition or multiplication when you press the right keys.

The software tells the computer what to do step by step. Even the largest and most expensive computer is like the comic character who needs directions for every step, no matter how simple or obvious it may be. The computer does not remember, learn from the past, or plan for the future. Instead it executes its current program mechanically, moving from one step to the next.

Again, a computer is much like a pocket calculator (see Figure 1-18). The calculator simply does what it is told. It does not improve with practice, remember steps, check to see that you have entered the data correctly, or jump ahead. The calculator is smart in the sense that it can do complex arithmetic in a few seconds, but it really isn't very bright.

How do computers guide spacecraft, control robots, answer questions, and play chess? Surely these tasks require intelligence, but the intelligence is in the software, not in the computer. It is clever work by the human programmer that leads the computer one step at a time through complicated tasks. Newcomers to the computer field are often surprised to discover how mechanically computers operate and how dramatically simple changes in their programs can affect their behavior.

Although computer hardware has become far less expensive and far more powerful than it was, computer software has not advanced as quickly. Writing computer programs—like composing music, writing books, or drawing pictures—is a skill poorly defined and difficult to teach. Creating computer software will be one of the great challenges of the future.

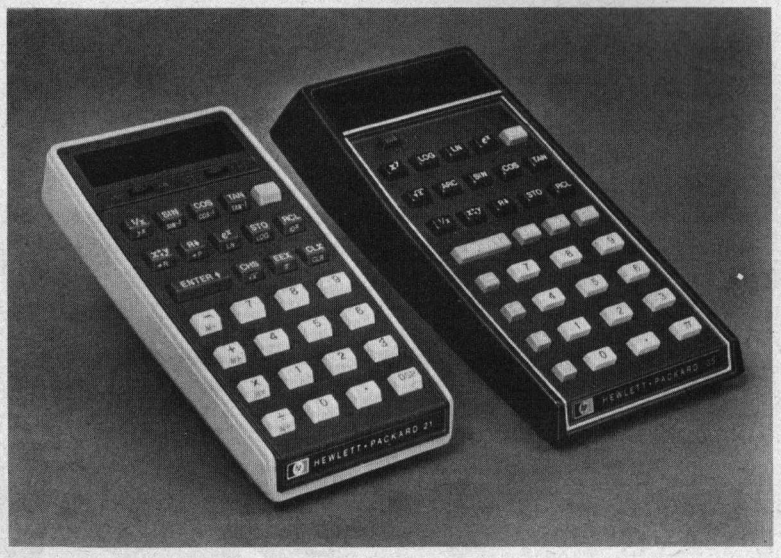


Figure 1-18. Two popular pocket calculators (Courtesy of Hewlett-Packard Company, Palo Alto, CA)

COMPUTERS AND MATHEMATICS

One reason why many people find a computer frightening is their belief that a computer is mathematical by nature. These people often assume that all computer programmers are mathematical geniuses who can solve long equations with no difficulty. In fact, most computer programmers do little mathematics. The fastest-growing applications of computers are in such areas as word processing, business data processing, computer-aided instruction, and communications networks (e.g., telephone systems). These applications seldom involve any mathematics beyond simple arithmetic. There is far less growth in the scientific use of computers that does involve higher mathematics.

Why, then, do most people associate computers with higher mathematics? The reason lies in the origin of computers. They were originally developed to solve complex mathematical problems such as aircraft design and missile guidance. Computers were first used as giant calculators and continue to be so used by many scientists and engineers. But a computer is not a calculator. Unlike a calculator, which can handle only numbers, a computer can handle any kind of information: numbers, typed characters, musical notes, geometric patterns, colors, textures, tastes, directions, or smells. The computer is a manager and manipulator of information, not a mathematician.

In fact, most programmers do not use calculus, algebra, trigonometry, or even geometry. Most computer problems involve logic rather than mathematics. So you don't have to be an expert in integral calculus or homological algebra to program a computer. Nor do you have to be able to write long formulas on a blackboard. The keys to working with computers are the ability to think logically and the patience to translate your thoughts into a computer program.

HOW DOES THE COMPUTER WORK?

The computer operates continuously and automatically from the time it is turned on until the time it is turned off. First, it fetches an instruction from its memory. Next, it does what that instruction tells it to do. Then, it fetches another instruction and so on, much like a person who is following a list of orders.

What are these instructions and where does the computer find them? Most instructions tell the computer to perform simple operations (like addition, subtraction, or multiplication) or to move data from one place to another. The computer fetches and executes one instruction at a time; it then proceeds to the next instruction at the next higher memory address unless the instruction tells it to do otherwise. If, for example, the computer has executed an ADD instruction that was stored in memory address 1000, it will then execute the instruction stored in memory address 1001,then 1002, and so on. The computer moves consecutively through its memory unless told to do otherwise, much as a person reads down a page unless directed to turn his or her attention somewhere else.

Special instructions, called jumps, stop the computer from moving straight ahead, just as special directions (e.g., "continued on p. 150") divert a reader. A jump instruction tells the computer specifically where to find the next instruction. For example, JUMP 2000 means "fetch the next instruction from memory address 2000." In Figure 1-19, the program is organized so that the computer starts at address 1000, proceeds consecutively to address 1500 and there finds the instruction JUMP 1000 which sends it back to address 1000. The computer will repeat this program until it is turned off.

How do the instructions get into the memory? The answer is that someone must put them there. For example, one could buy a cassette tape containing a program and play the cassette tape into the computer's memory. The computer would then be able to execute the program.

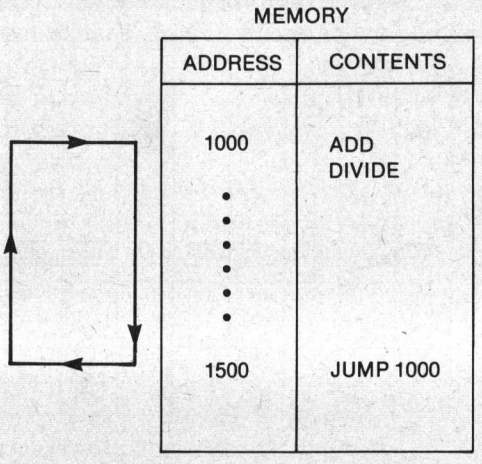


Figure 1-19. The use of a JUMP instruction. The computer executes instructions consecutively until it reaches the JUMP instruction that sends it back to address 1000.

How do we start the computer? Usually the computer has a key or button marked RESET. Pressing RESET forces the computer to start executing instructions at a particular address in its memory, usually address zero. Now all the operator has to do is load the program into memory starting at address zero, press RESET, and watch the program run.

A computer operates automatically without thinking or observing the world around it; it just fetches and executes instructions. A computer does not learn or remember. It will only execute the program currently in its memory; it will not remember an old program or greet that program with any familiarity at a later time. The computer simply does what it is told, much like a student who memorizes a textbook for an examination and then immediately forgets all that he or she has learned.

HOW CAN COMPUTERS SEEM TO BE SO SMART?

If that's all computers do, how can they seem to be smart? How can a computer learn a person's name, solve a mathematical problem, or provide information about accounts, criminals, or credit ratings?

The answer is that the program provides the intelligence. For example, let's see how the computer types a message like "HOW ARE YOU?" The programmer must first place the message in memory (including the spaces and the question mark) and also place in memory a program that takes characters one by one and types them. Remember that the program is just a list of instructions for the computer to follow. In our example, the program must count the number of characters the computer has already typed and keep track of where the next character is located in memory. Each time the computer takes a character from memory, it moves on to the next character at the next higher address, just as a typewriter moves to the next position on a line after you type a character. Figure 1-20 illustrates the procedure showing the contents of memory, the addresses, and the number of characters the computer has typed when it finishes with a particular address. Note that the spaces and the ques-

MEMORY CONTENTS	H	O	W		A	R	E		Y	O	U	?
MEMORY ADDRESS	100	101	102	103	104	105	106	107	108	109	110	111
NUMBER OF CHARACTERS TYPED	1	2	3	4	5	6	7	8	9	10	11	12

Figure 1-20. How the computer types a message. The program tells the computer to type the 12 characters that are in its memory starting at address 100, much as one could tell a sanitation crew to pick up the garbage from 12 houses on a street starting at a specific address.

tion mark must also be loaded into the memory and typed by the computer, just as you would have to remember to type them on a normal typewriter.

We can describe the program in steps as follows:

STEP 1

NUMBER OF CHARACTERS TYPED = 0
ADDRESS OF NEXT CHARACTER TO BE TYPED = 100 (since the message starts in memory address 100)

STEP 2

TYPE THE NEXT CHARACTER

STEP 3

ADD 1 TO NUMBER OF CHARACTERS TYPED

PROCEED TO THE NEXT ADDRESS (I.E., ADD 1 TO THE ADDRESS OF NEXT CHARACTER TO BE TYPED)

STEP 4

IF 12 CHARACTERS HAVE NOT BEEN TYPED, GO BACK TO STEP 2

(The limit is 12, since the message consists of 12 characters, including the spaces and the question mark.)

What we have told the computer to do is "Type 12 characters starting with the one in memory address 100." Figure 1-21 contains a flowchart of the procedure. The program tells the computer how many characters to type and where to find them. The computer does not examine the characters or understand them. Nor can it type any messages that the programmer has not placed in its memory. In fact, if the programmer makes a spelling error or forgets a space when entering the message, the computer will make the same mistake.

We can use a similar program to have the computer read what you type and repeat it (like a parrot). Let us assume that the computer has just typed the message "WHAT IS YOUR NAME?" Now the next program that it executes will take the name that you enter and save it in memory. After you enter your name, you must press the carriage return key to tell the computer that you are finished.

What we must tell the computer to do is "place the typed characters in memory starting at address 200 and quit when the typist presses the carriage return." We can describe this program as follows (here Figure 1-22 illustrates the procedure and Figure 1-23 contains the flowchart):

STEP 1

ADDRESS IN WHICH TO STORE NEXT CHARACTER = 200
(since we will start saving characters in address 200)

STEP 2

STOP IF THE TYPIST PRESSES THE CARRIAGE RETURN;
OTHERWISE, READ THE NEXT CHARACTER AND SAVE IT IN MEMORY

STEP 3

PROCEED TO THE NEXT ADDRESS (I.E., ADD 1 TO THE ADDRESS IN WHICH TO STORE NEXT CHARACTER)
RETURN TO STEP 2

The computer simply takes the characters that you type and saves them in its memory. It does not understand the characters nor does it examine the name. If you enter CINDER ELLA or NEW ORLEANS, it will accept that as your real name. The computer does not recognize impossible combinations of letters (such as QZXY), words (even four-letter ones), jokes, or abbreviations. The computer can now amaze you by typing your name; all it has to do is execute our typing program starting at memory address 200 and continuing through the number of characters that you entered.

So if you enter your name as JOE SCHMOE, the computer can type the message HELLO THERE, JOE SCHMOE, HOW ARE YOU? as follows (assuming that the programmer has placed the appropriate characters in memory):

Type 13 characters starting at address 100. The thirteen characters are:

the word HELLO
the space between HELLO and THERE
the word THERE
the comma after THERE
the space after the comma

Type 10 characters starting at address 200. These ten characters are the ones that you entered as your name (including the space between your first name and your last name).

Type 14 characters starting at address 400. The fourteen characters are:

the period after your name
the space after the period
the word HOW
the space between HOW and ARE
the word ARE
the space between ARE and YOU
the word YOU
the question mark

The computer simply does what the programmer tells it to do. It does not recognize words, grammar, syntax, punctuation, or spaces. Remember that this lack of recognition means that the computer can be equally adept at any language or dialect. However, the computer will not recognize or correct mistakes.

One question that we didn't answer is how the computer recognizes the carriage return. In fact, the computer handles all characters as numbers. When you strike a key on a keyboard, the computer receives a particular number. For example an "A" on the keyboard might be "65" to the computer, a "B" might be "66," and so on. This correspondence is just like Morse Code. In fact, most personal computers and their peripherals use a code called ASCII (pronounced "Ass-kee" and defined in Appendix 2). Note, as Figure 1-24 shows,

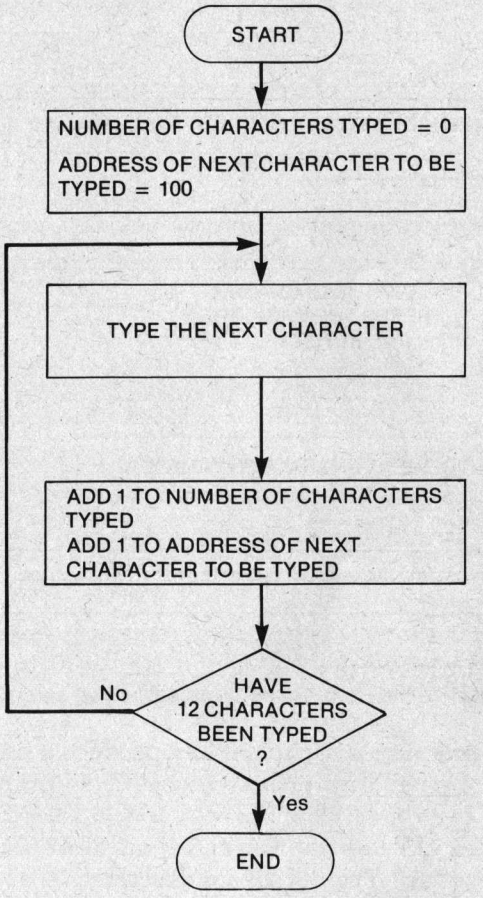

START
NUMBER OF CHARACTERS TYPED = 0
ADDRESS OF NEXT CHARACTER TO BE TYPED = 100
TYPE THE NEXT CHARACTER
ADD 1 TO NUMBER OF CHARACTERS TYPED
ADD 1 TO ADDRESS OF NEXT CHARACTER TO BE TYPED
HAVE 12 CHARACTERS BEEN TYPED ?
No
Yes
END

Figure 1-21. Flowchart of the procedure for typing a message

MEMORY CONTENTS	J	O	E		S	M	I	T	H
MEMORY ADDRESS	200	201	202	203	204	205	206	207	208
NUMBER OF CHARACTERS REMEMBERED	1	2	3	4	5	6	7	8	9

Figure 1-22. How the computer remembers a name. The program tells the computer to save the typed characters in its memory starting at address 200 until the operator types a carriage return, much as one could tell a delivery service to leave circulars at each house on a street starting at a specific address and ending at the next corner.

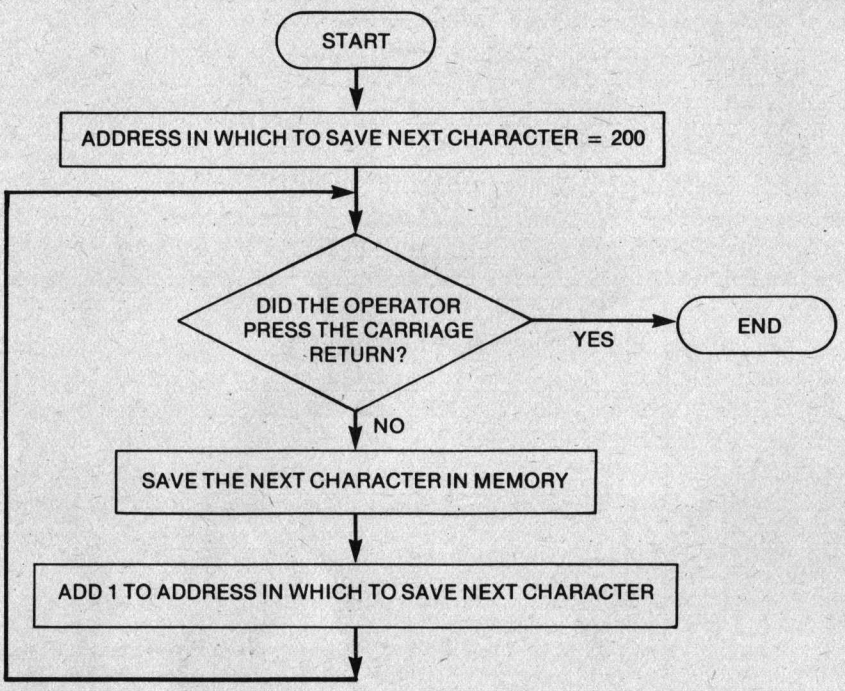

START
ADDRESS IN WHICH TO SAVE NEXT CHARACTER = 200
DID THE OPERATOR PRESS THE CARRIAGE RETURN?
YES
END
NO
SAVE THE NEXT CHARACTER IN MEMORY
ADD 1 TO ADDRESS IN WHICH TO SAVE NEXT CHARACTER

Figure 1-23. Flowchart of the procedure for remembering a name

that not only does pressing a particular key produce a particular number but also sending that number to a printer causes it to print the corresponding letter.

What about spaces, question marks, and commas? Furthermore, how about a carriage return or a tab? These, too, are characters that have corresponding codes. For example, a space might be 32, a decimal point 46, and a carriage return 13. When you press the space bar, the computer receives a 32. When the computer sends the printer a 13, the printer performs a carriage return. The computer doesn't know which numbers mean letters, which mean digits, which mean punctuation marks, and which mean operations such as backspace and carriage return.

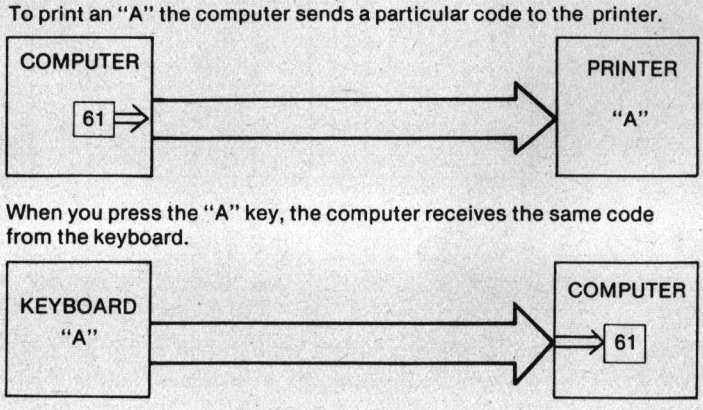


Figure 1-24. The use of character codes

The important point here is that the computer does not understand the sigficance of the words it prints or reads or speaks, any more than does a television set, typewriter, or tape recorder. The computer merely does what it is told. The programmer must be sure that the instructions are clear, unambiguous, and sensible.

The computer's lack of insight and experience means that it has no built-in preferences, no inherent language, no cultural or social bias, and no prejudices. It also, of course, has no morals, no shame, and no regard for feelings. Its messages can be surly, humorous, or even obscene, depending on the choice of the programmer.

HOW DO COMPUTERS DIFFER?

All computers have the same basic structures, methods, and abilities. Why, then, are some computers better than others? Here are some of the factors that make computers different.

Price

Obviously, some computers are more expensive than others. The computer in Figure 1-25 typically costs several million dollars while the computer in Figure 1-26 costs only a few hundred dollars. We may roughly categorize computers as follows (see Figure 1-27):

—A computer costing over $100,000 is a large computer or *mainframe.* Well-known manufacturers of mainframes include IBM, Honeywell, Sperry Univac, Burroughs, Control Data, and NCR. Such computers have been produced since about 1950.

—A computer costing between $10,000 and $100,000 is a small computer or *minicomputer.* Well-known manufacturers of minicomputers include Digital Equipment (DEC), Data General, and Hewlett-Packard. Minicomputers have been produced since about 1963.

—A computer costing less than $10,000 is a very small computer or *microcomputer.* Manufacturers of microcomputers include Apple, Atari, Commodore, Compucolor, Cromemco, Exidy, Heath, Ohio Scientific Instruments, Pertec, Radio Shack, Southwest Technical Products, and Texas Instruments. Microcomputers have been produced since about 1975.

Speed

Some computers work faster than others.

—*Mainframes* can execute as many as 10,000,000 instructions per second.

—*Minicomputers* can execute as many as 1,000,000 instructions per second.

—*Microcomputers* can execute as many as 250,000 instructions per second.

Figure 1-25. The IBM 370/168, a popular mainframe computer (Courtesy of IBM Corp., Armonk, NY)

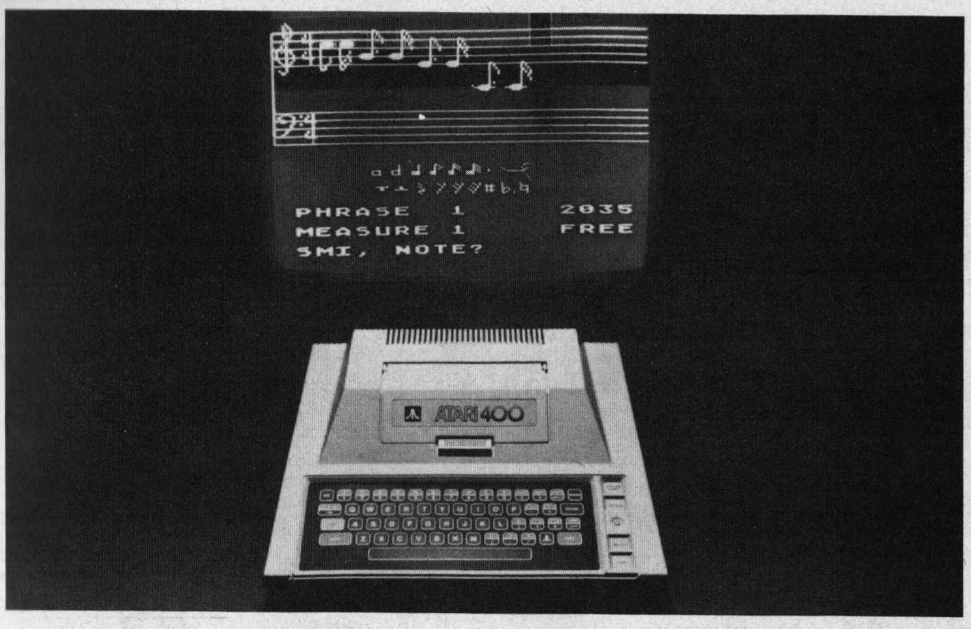


Figure 1-26. The Atari 400 Personal Computer System (Courtesy of Atari, Inc., Sunnyvale, CA)

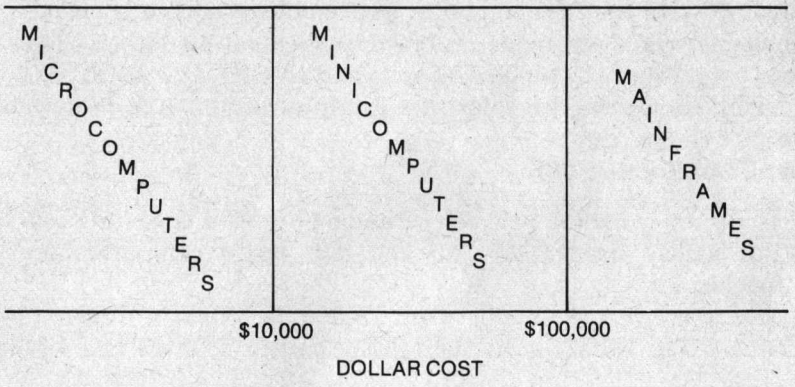


Figure 1-27. Costs of computers. A fully equipped microcomputer can cost as much as $20,000, and a fully equipped minicomputer can cost $250,000.

So a mainframe typically works ten times as fast as a minicomputer and forty times as fast as a microcomputer. Note that all computers are incredibly fast. By comparison, a person's response time is about a tenth of a second and a calculator can execute about 1,000 instructions per second. Thus the high price of the mainframe and minicomputer bring the purchaser higher speed. Figure 1-28 compares the speed of different types of computers.

Speed is a major problem in some applications. For example, a flight simulator for training pilots must present sounds, visual effects, motion effects and other stimuli at the same rate as they occur in the real world. Otherwise the simulator will not prepare the trainee to fly an actual airplane. We say that the simulator must operate in *real time* since it must produce events at the same rate at which they would really happen. Other real-time tasks are missile guidance and control of oil refineries and machine tools. Tasks like

payroll preparation, accounting, computer-aided instruction, and playing chess, are not real-time, since the time at which the answers appear is not critical. Large, fast computers can handle real-time tasks far more effectively than small, slow computers can.

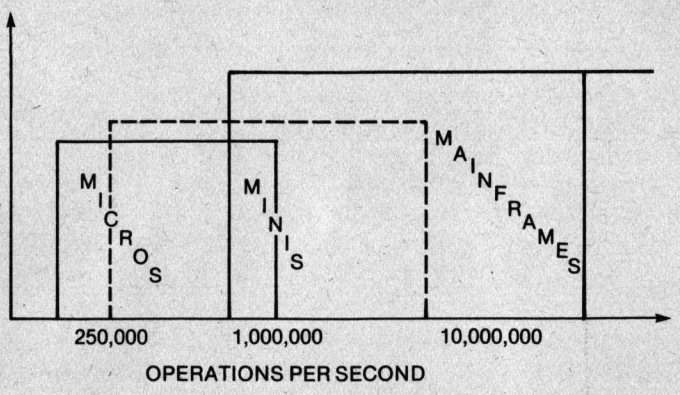


Figure 1-28. Speeds of computers

Word Length

The amount of work a computer can do depends not only on how fast it can perform operations but also on how much data it can handle in each operation. The amount of data that the computer can handle at a time is called a *word.* The computer's words are made up of *bits* which can be either ON or OFF. Sometimes we refer to ON as TRUE or 1, and to OFF as FALSE or 0. Since bits are such small units, we refer to a group of bits as a *byte* (pronounced "bite" despite the spelling).

In general (see Figure 1-29),

—*Mainframes* handle between 32 and 64 bits at a time (or 4 to 8 bytes). The widely used IBM 360 and 370 series computers are 32-bit machines.

—*Minicomputers* handle between 12 and 32 bits at a time (or 1½ to 4 bytes). The widely used Digital Equipment PDP-11 is a 16-bit computer.

—*Microcomputers* handle between 4 and 16 bits at a time (or ½ to 2 bytes). The popular Apple II, Commodore PET, and Radio Shack TRS-80 are 8-bit computers.

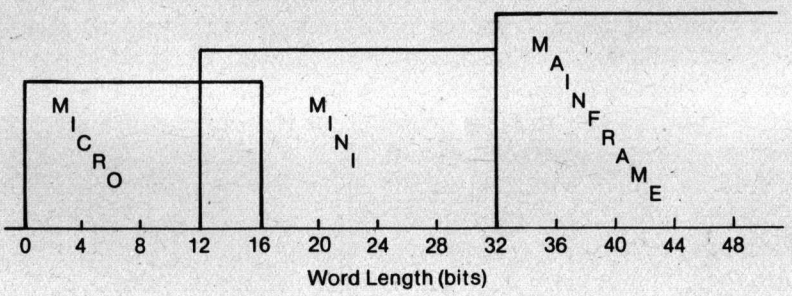


Figure 1-29. Comparison of computer word lengths

Mainframes and minicomputers not only work faster than microcomputers, but they also handle more data at one time. For example, a 48-bit mainframe that performs 5,000,000 instructions per second is really 120 times as powerful as an 8-bit microcomputer that performs 250,000 instructions per second since the mainframe handles 6 times as much data in one-twentieth the time. But keep in mind that all computers are incredibly fast by ordinary standards.

Peripherals and Software

The cost and capabilities of a computer obviously depend on how many peripherals and how much software it has. Even more important than sheer numbers are the capabilities of the peripherals and software, including how much data they can handle, what tasks they can perform, and how fast they can operate.

Peripherals and software present some special problems: They cost just as much to build or develop for small computers as for large computers. Clearly the cost of a typewriter or card reader does not depend on the size or cost of the computer to which it is attached. Nor does the time and skill required to write a program depend on the size or cost of the computer on which it will run. Thus the cost of software and peripherals may easily dwarf the cost of a microcomputer; users of microcomputers may find themselves in the position of a tourist who can obtain a new discount airfare to a resort, but then finds that he still cannot afford the hotel or the meals.

They are far less likely to be available for small computers than for large computers. Developing programs and peripherals for a particular computer takes time and money. Small computers have not been produced for as long as have larger computers, and most manufacturers of small computers are relatively small companies with limited resources. Purchasers of microcomputers may find themselves in the position of a person who buys the newest kind of sound system, but then finds that there are no speakers to attach to it or recordings to play on it.

Obviously time and the increasing size of the personal computer market will solve some of these problems. Peripherals will be available for popular computers at reasonable prices (the high volume of sales will reduce costs somewhat). Mass distribution can reduce the cost of software substantially. If retailers can sell thousands of copies of the same program, then its originators can make a profit at a relatively low per-unit price. When the number of potential customers becomes large enough, popular computer programs should cost the same as popular books or records. The cost and availability of peripherals and software, however, will be a major problem for users of personal computers for years to come.

Uses

Mainframes, minicomputers, and microcomputers are no more competitive than are airplanes, trucks, and cars. They offer different amounts of computing power at different prices. The number of small computers, of course, has already greatly exceeded the number of mainframes and minicomputers and will do so even more in the future.

Certain problems, however, require a mainframe. They include large data processing tasks, such as printing telephone bills or controlling the inventory of a chain of department stores. They also include large scientific, business, and engineering problems such as forecasting the weather, predicting the course of a national economy, or designing a new missile or airplane.

Minicomputers can handle less demanding versions of the same problems as well as other applications that require a medium-sized computer. Such applications include library management, banking terminals, automated warehouses, large medical instruments, and typesetting equipment.

Microcomputers fit into still smaller applications, such as handling inventory or accounts for a small business, controlling a video game or an electronic cash register, or keeping track of records for a bowling league or a church. The microcomputer, although it cannot replace larger computers, can bring computing power into the home or small business, thus simplifying many common tasks as well as providing education and entertainment. The major question that remains to be solved is how long it will take to develop the software, educational materials, and basic familiarity that will allow users to take advantage of personal computers.

REFERENCES

Bylinsky, G. "Here Comes the Second Computer Revolution." *Fortune,* November 1975, p. 134.

Bylinsky, G. "The Computer Stores Have Arrived." *Fortune,* April 24, 1978.

Davis, W. S., and McCormack, A. *The Information Age.* Reading, MA: Addison-Wesley, 1979.

Frenzel, L. *Getting Acquainted with Microcomputers.* Indianapolis: Howard W. Sams, 1978.

Libes, S. *Small Computer Systems Handbook.* Rochelle Park, NJ: Hayden, 1978.

Wels, B. *Personal Computers.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

Willis, J. *Peanut Butter and Jelly Guide to Computers.* Forest Grove, OR: Dilithium Press, 1978.

2 Components of a Computer

This chapter describes the hardware and software needed to form a useful computer. We stress items used in personal computers although their functions are essential in any computer. We do the following.

—List the requirements for a useful computer.
—Discuss the major categories of personal computers.
—Describe what the various components do.
—Define terms frequently used in describing components.
—Discuss different versions of components.

Personal computers vary greatly in cost, size, features, and applications. The computer in Figure 2-1, for example, costs only a few hundred dollars, but its exposed form and limited peripherals make it more useful for an engineer than for a mathematics teacher or accountant. On the other hand the computer in Figure 2-2, together with peripherals and software, costs several thousand dollars; such a computer is more than most hobbyists or classroom

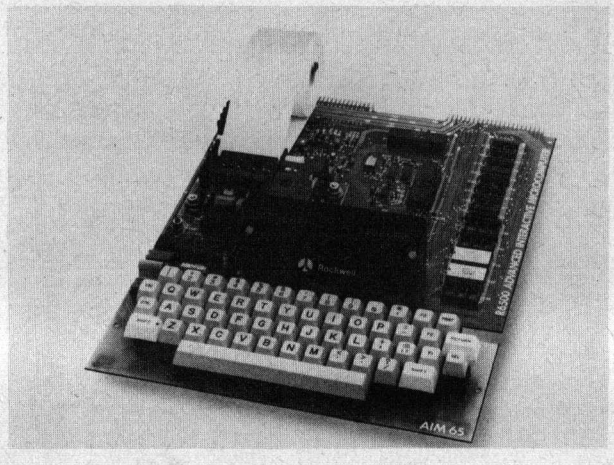


Figure 2-1. Rockwell AIM-65, the Advanced Interactive Microcomputer with on-board printer (Courtesy of Rockwell International, Electronic Systems Division, Anaheim, CA)

Figure 2-2. The Vector Graphic Vector 1 8080-based microcomputer (Courtesy of Vector Graphic, Inc., Westlake Village, CA)

teachers could afford, but may be well-suited to a small business, professional office, or scientific laboratory. In this chapter, we describe the common requirements and components of such computers; we also divide personal computers into their major categories and discuss their specific characteristics. In Chapter 5 we describe peripherals in more detail and in Chapter 8 we discuss the choice of a computer.

HARDWARE

A useful computer must include the following hardware:

—Central processing unit or CPU
—Memory
—Input peripherals and connections to the computer (or *interfaces*)
—Output peripherals and interfaces
—Mass storage devices and interfaces
—Incidentals such as a power supply, a case, connectors, and cables (We will not discuss these, although they are always necessary.)

The central processing unit is the active, "thinking" part of a computer. Although the CPU (see Figure 2-3) has been made small and inexpensive, it still establishes the entire personality of the computer, much as the brain does for a person. The memory stores programs and data. The more memory a computer has, the more data and programs it can have readily available. The standard unit of memory is 1024 bytes or 1 *kilobyte* (like kilometer), usually abbreviated as 1 KB or 1 K.

Today's computers have two types of memory:

—*Read-only memory (ROM),* which cannot be changed in normal operation. You can think of ROM as a "stone tablet" in which data or programs are stored permanently. The contents of ROM do not change when you turn the computer off.
—*Read/write memory (RAM),* which can be changed. This is ordinary memory used to store data and programs temporarily. However, RAM loses its contents when you turn the computer off; this characteristic is known as *volatility*.

Read-only memory helps make personal computers easy to use. Programs in read-only memory need not be loaded each time the computer is turned on. Thus a computer with programs in ROM retains its basic skills; it does not have to be completely re-educated every time you start it. If the computer does not have programs in ROM, the user is like a teacher whose students forget everything they have learned overnight. Clearly, just getting back to where you were on the previous day is a major achievement. Furthermore, you need a way to get started (a bootstrap), since the computer has forgotten everything it ever knew and is as difficult to teach as a newborn baby.

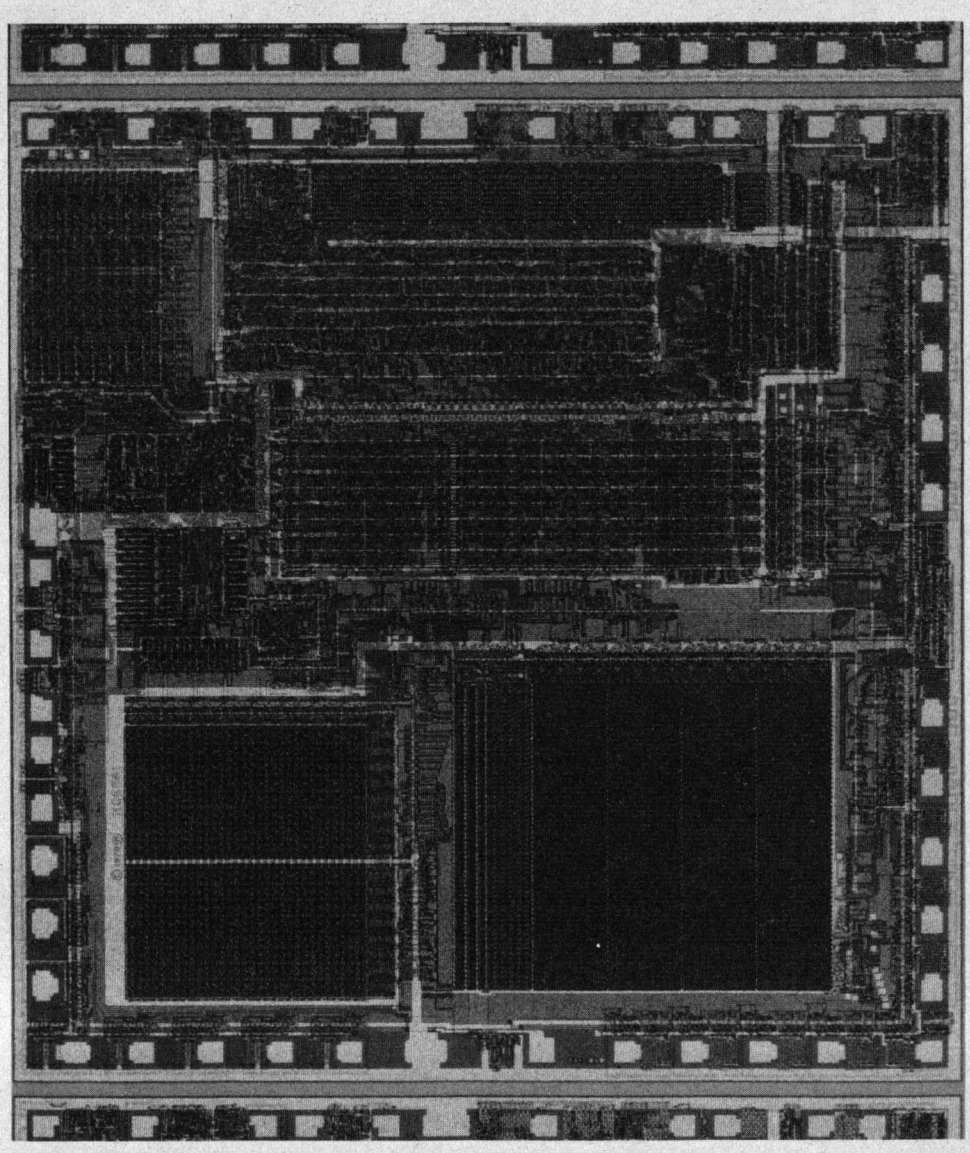


Figure 2-3. Photomicrograph of the Motorola 6801 microprocessor. The actual device is less than a quarter of an inch long. (Courtesy of Motorola Semiconductor Products, Inc., Phoenix, AZ)

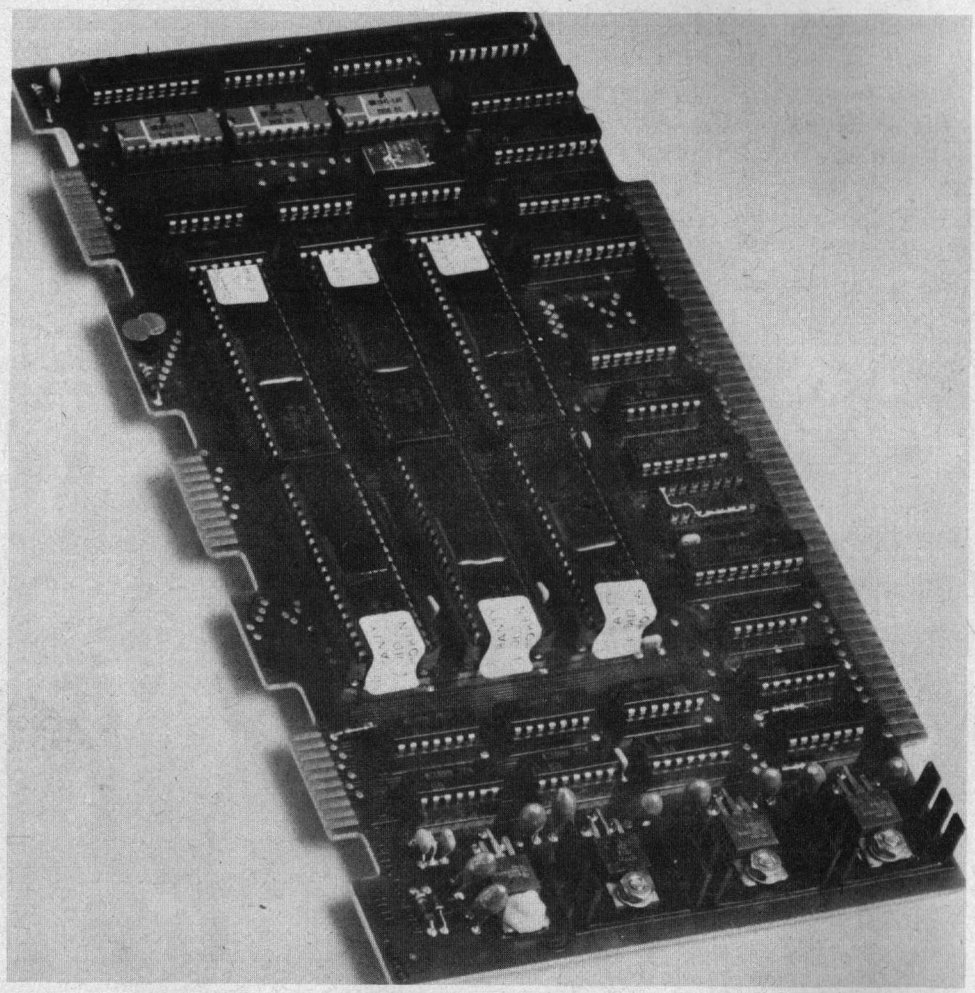


Figure 2-4. A typical interface board, the Alpha Micro Serial I/O Interface. Note the numerous circuits and other components that are required. (Courtesy of Alpha Micro, Inc., Irvine, CA)

Input and output peripherals (see Chapter 5 for more details) include keyboards, typewriters, card readers, printers, and video displays. The computer does not instinctively know how to use these devices. Just like any other task, operating a peripheral requires special hardware and software. The hardware is called an *interface* (Figure 2-4 shows an example) and the software is called an *input/output (I/O) driver.* We discuss both at greater length in Chapter 6.

Mass storage is like the files in a business, the records and tapes in a stereophonic system, or the books in a library. This is where programs and data are saved for later use. Popular media for mass storage include punched paper tape, cassettes, and disks. As with audio cassettes or tapes, you may buy media blank or with programs or data on them. The media themselves may be saved anywhere. Only the ones currently in use have to be immediately available to the computer. In Chapter 5 we describe mass storage in more detail.

SOFTWARE

A computer must have software (i.e., programs) if it is to do anything. We may compare software to a football team's playbook. The team may have many outstanding players, but it cannot accomplish anything without a set of plays that tell the players where to line up, when to start, which direction to move, and whom they are supposed to block or tackle.

We also need a form in which to describe what we want the computer to do, much as a composer needs special notation to describe a piece of music and a football coach needs a set of symbols (the famous X's and O's) with which to describe a play. The form in which we describe procedures for the computer to follow is called a *computer language.* The computer itself directly understands (and can execute) only numerical codes that denote operations. These codes are unique to a particular CPU; for example, one CPU might interpret the number 11 to mean ADD, the number 25 SUBTRACT, and the number 43 STORE DATA IN MEMORY. The manufacturer of the CPU assigns these numbers arbitrarily, much as the telephone company assigns area codes—212 to New York City, 213 to Los Angeles, and 312 to Chicago. A programmer can write programs as a series of numbers (called *machine language*), but this method is slow and tedious. Furthermore, such programs are very difficult to correct or modify.

Assembly language provides a simple solution to this problem. It assigns meaningful words or abbreviations (still unique to a particular CPU) to the numerical codes (e.g., ADD for addition instead of 11, SUB for subtraction instead of 25, and STO for storing data instead of 43). Clearly a program consisting of words or abbreviations is easier to write and modify than one consisting of arbitrary numbers. However, someone (either a person or a computer) must still convert the words or abbreviations into the numbers that the computer understands.

More advanced computer languages (called high-level languages) make programming even simpler, because they allow the programmer to tell the computer what to do without specifying precisely how the computer is to do it. The programmer working in a high-level language can write a statement like "ADD MONTHLY SALES TO TOTAL SALES". A translator (again either a person or a computer) must convert the statement into the detailed series of instructions that the computer actually executes. Most computer users find high-level languages (popular ones are BASIC, COBOL, FORTRAN, and PASCAL) far easier to use and understand than the more detailed assembly and machine languages.

We commonly divide programs into two types.

—Systems software that helps you use the computer (see Figure 2-5)
—Applications software that actually performs the work

Systems programs are like white-collar workers who administer, operate, and service the computer while applications programs are like blue-collar workers who turn out the products. The same systems programs may be used in many different applications, such as financial planning, game playing, or word processing, much as a white-collar worker, such as an accountant or an advertising manager, may move from one industry to another.

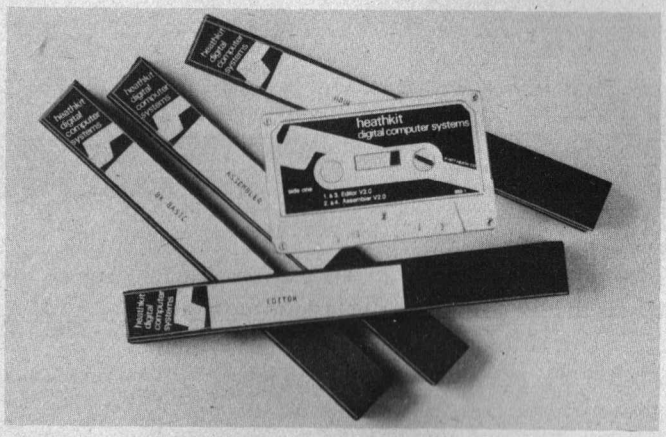


Figure 2-5. Systems software available on paper tape or cassette for the Heath H-8 computer. Included are an assembler, editor, debugger, and an 8K BASIC interpreter. (Courtesy of Heath Co., Benton Harbor, MI)

The most important systems programs are

—*The operating system (OS) or monitor* controls overall operations. This program is the chief executive; it is in charge when applications programs are not running. It lets the user communicate with the computer and run other systems programs and applications programs.

—*Peripheral or I/O drivers* handle communications with peripherals, providing the proper flow of data in the correct form to and from the computer.

—*Assemblers, compilers, and interpreters* translate programs written in forms that people can understand into the form that the computer can actually execute. The input may look like ordinary sentences or may consist of mathematical symbols or other notations that are commonly used in particular applications.

—*Editors* allow the user to prepare printed matter or input for other programs. The input may itself be a computer program that must be translated; it may also consist of numbers or words for use in headings or in computer-generated letters or documents. Editors specifically intended to produce letters or documents are called *word processing systems.* Typical tasks that an editor can perform are inserting, deleting, or replacing characters and adding or removing lines.

—*Debuggers* help the user find errors in a program. Debuggers can display intermediate results, stop the program under specified conditions, execute the program one step at a time, and show what is happening inside the computer.

Applications programs do the actual work but systems programs must control, correct update, and translate the applications programs. Ultimately, however, it is the applications program that enters invoices into a business ledger, prints payroll checks, controls a laboratory instrument, or provides an arithmetic quiz for a student.

DATA BASES

Programs only tell the computer what to do; the computer must also have data to manipulate. In some cases, such as an arithmetic lesson or the calculation of a loan payment, the data is just a few numbers that the operator can enter by hand. Similarly the results of such tasks are a few numbers that can be saved on a piece of paper.

Other tasks require a large amount of well-organized data. For example, the records of a small business may contain entries for hundreds of customers, employees, suppliers, transactions, and inventory items. Certainly no one would want to enter these records into a computer by hand every time they were needed. Rather, one wants to enter the information once, save it, and update it regularly. Such a collection of data is called a *data base.*

Sometimes you can purchase data bases or obtain information from them by telephone. An investor, for example, may want a data base containing the current prices of stocks, bonds, or commodities. A realtor may want a data base that lists all properties for sale in the local area. Researchers, government agencies, and advertisers may want data bases containing lists of members of organizations, purchasers of new homes, parents of school-age children, or college graduates.

One problem is that the various users often want different information from the data base. For example, consider a file containing invoices for a business. A salesman may want to know which customers have purchased a particular item or are located in a particular city. The accounting department may want to know which accounts are overdue and by how long. An order clerk may want to know which customers have unfilled orders. Other useful information might be total dollar volume by account, by item, and by month; trends in volume by customer and by item; and past payment and order records.

In the case of the stock market, one user may want to know the fluctuations in the value of a particular stock or of all the stocks in a particular industry. Another user may want to identify stocks that yield high dividends or have low price/earnings ratios. Still another user may want to know which stocks have led or trailed the market averages. Another user may have a particular portfolio of stocks to follow.

To get a variety of information from a data base requires a program called a *data base manager* or *data base management system.* The user can make inquiries or *queries* for data, often in a special way that makes it easy to specify the required information.

Many programs may use the same data base. A payroll program may use employee records to print checks. A tax reporting program may use the same records to calculate tax payments. An updating program may need to change data and add or delete employee records. A mailing list program may use the same records to address forms or notices.

COMPATIBILITY

Not only does a computer need hardware, software, and data to accomplish anything, but it also must have them in compatible forms. The hardware, the software, and the data must all fit together, much as the speakers, amplifiers,

turntables, and phonograph records must all fit together to produce a working stereophonic system. Unfortunately computers are far less standardized than are records, tapes, players, and recorders. Most programs will only work on a particular computer with a particular set of peripherals and data organized in a particular way.

Not only must the hardware, software, and data be compatible, but their components must also be compatible. The various parts of the hardware must work together as must the various programs. The lack of industry-wide standards can make this compatibility surprisingly difficult to achieve. Computers, therefore, are not interchangeable. Furthermore, owning a computer does not necessarily mean that one can use existing programs, peripherals, and data bases.

Not only must the various parts of the computer work together, but they must also be compatible from the user's point of view. The operating procedures for all peripherals and programs should be as similar as possible. The user quickly becomes confused when a letter means one thing in one program ("D" means delete), another thing in another program ("D" means double-spacing), and still others in other programs ("D" means displaying the memory or moving a line down on the screen). The user also becomes annoyed when one program requires precise spacing and punctuation, while another program regards that same spacing and punctuation as erroneous. The computer handles each task separately; it does not care about consistency. People, however, learn and remember; therefore they prefer that all procedures be similar.

CATEGORIES OF PERSONAL COMPUTERS

Personal computers can be divided into three general categories.

—One-board computers (see Figures 2-6 through 2-9)
—Appliance computers (see Figures 1-2, 1-4, 1-5, and 2-10 through 2-12)
—Modular computers (see Figures 1-1 and 2-13 through 2-16)

Some computers do not fit neatly into these categories, and expansion units can move a computer into a different category. However, we will concentrate on these categories and describe their features and applications. In Chapter 8 we use these three categories in discussing how to choose a computer.

ONE-BOARD COMPUTERS

One-board computers like those in Figures 2-6 through 2-9 are inexpensive, entry-level computers for electronics hobbyists and engineers. They were the first inexpensive personal computers, since they are easy to design and build. Many one-board computers have been described in hobby magazines and offered by semiconductor manufacturers to promote the use of new microprocessors. They are also popular for short courses that include a "take-home" computer. Leading manufacturers of one-board computers are E and L Instruments, Heath, Integrated Computer Systems, Intel, MOS Technology,

Figure 2-6. The Signetics Instructor 50 Computer System, a one-board computer with a simple case (Courtesy of Signetics Corp., Sunnyvale, CA)

Motorola, Netronics, Ohio Scientific, RCA, Rockwell, SD Systems, and Synertek.

These computers typically consist of a single unpackaged circuit board, although a few (like the Signetics Instructor 50 of Figure 2-6) have a simple case. The board usually contains

—Microprocessor CPU
—Read-only memory containing a simple operating system or *monitor.*
—A small amount of read/write memory (typically 1K or less) for storing data and programs
—Input/output ports and an edge connector to which simple peripherals (such as a small motor) can be attached
—A small, calculator-like keyboard that can be used to enter numbers and to operate the computer
—Displays like those in pocket calculators.

Some one-board computers have such additional features as:

—Audio cassette interface
—Teletypewriter interface
—Space for on-board ROM for completed user programs
—Television or video interface
—Small thermal printer like the ones used in calculators and cash registers
—Larger and better displays.

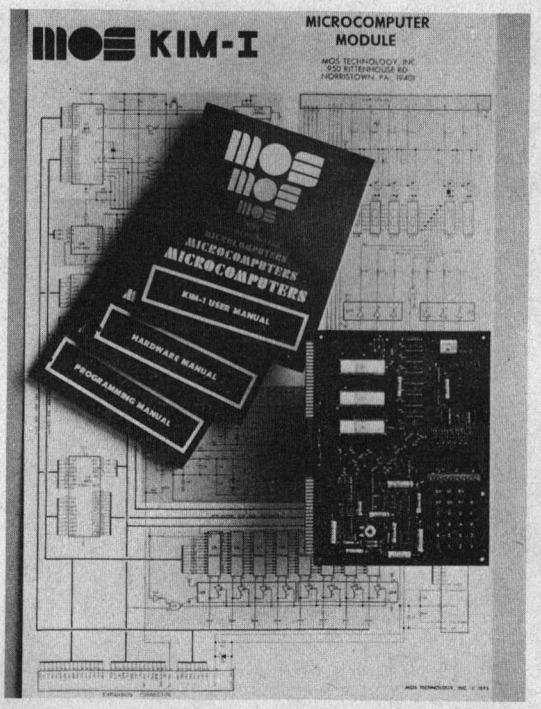


Figure 2-7. The MOS Technology KIM one-board computer, currently the most popular entry in its category (Courtesy of MOS Technology, Inc., Norristown, PA)

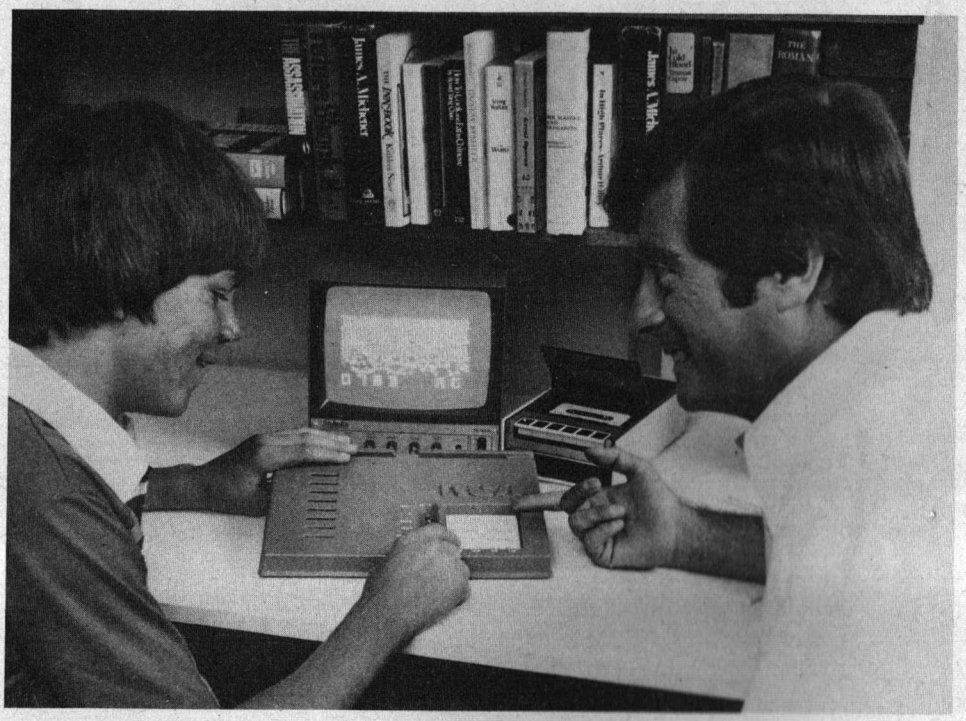


Figure 2-8. The RCA COSMAC VIP personal computer with an attached television set and cassette recorder (Courtesy of RCA, Lancaster, PA)

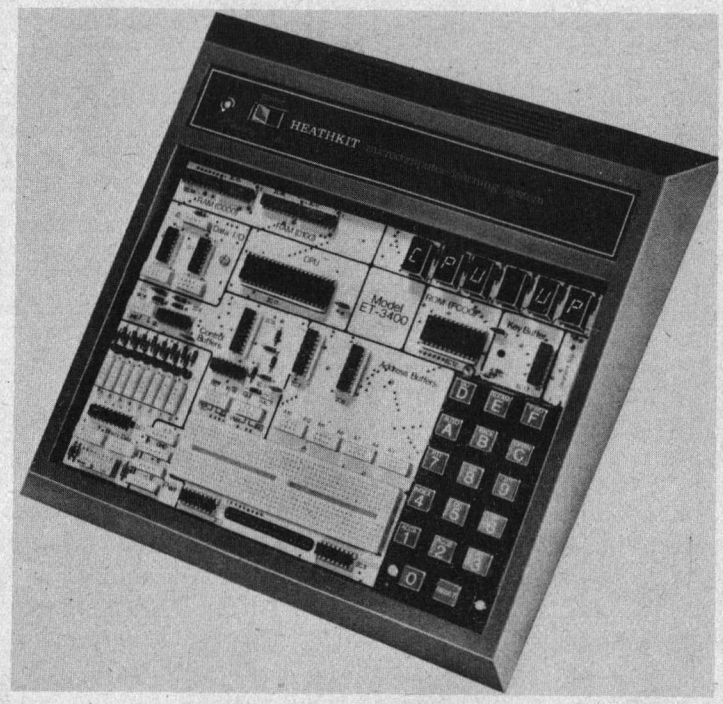


Figure 2-9. The Heath ET3400 computer trainer (Courtesy of Heath Co., Benton Harbor, MI)

Most one-board computers do not include a power supply and some require a substantial amount of assembly. Some come without peripherals and depend on an external keyboard and television set.

The MOS Technology KIM (Fig. 2-7) is the most popular one-board computer. It was the first fully assembled, self-contained computer to be widely available at an affordable price (originally about $250 without a power supply). Thousands of KIMs have been sold and there are numerous books, courses, publications, hardware and software suppliers, and user's groups dedicated to the KIM and related computers such as the Synertek SYM and Rockwell AIM-65 (Figure 2-1). Similar computers include the Netronics ELF, RCA COSMAC VIP (Figure 2-8), and the Heath ET3400 Microprocessor Trainer (Figure 2-9).

Many one-board computers have little documentation or support, since they are intended to allow engineers to evaluate a new microprocessor at low cost for potential use in products. Such computers are not intended for the novice or for the system builder.

Typical applications of one-board computers are

—Trainers for electronics courses
—Prototyping of products that contain microprocessors
—Dedicated controller for laboratory equipment
—Evaluation board for microprocessors
—Data logger that reads data from sensors and saves the values on cassette tape
—Terminal for entering small amounts of data into a remote computer.

APPLIANCE COMPUTERS

The appliance computer is the popular entry-level computer for those who are not electronics hobbyists or engineers. This category of computer is intended for teachers, businesspeople, programmers, scientists, and game players. Appliance computers are the most widely available and most popular personal computers. Leading manufacturers of appliance computers are Apple, Atari, Commodore, Compucolor, Exidy, Ohio Scientific, Radio Shack, and Texas Instruments.

A typical introductory system costs under $1,000 and includes

—A packaged computer with CPU and memory
—A typewriterlike keyboard
—Interfaces for a television set and cassette recorder

Figures 2-10, 2-11, and 2-12 show typical appliance computers; additional examples are in Figures 1-2, 1-4, and 1-5. The basic system may actually include the television set and cassette recorder. The computers in Figure 1-5, 2-10, 2-11, and 2-12 can use a color television; the others cannot.

Appliance computers come ready to use. All have an operating system, editor, debugger, and BASIC interpreter (see Chapter 3) in read-only memory so you can easily run the computer and write your own programs. The manufacturers and other suppliers also sell programs prerecorded on cassettes. Simple programs usually cost between five and twenty dollars, while

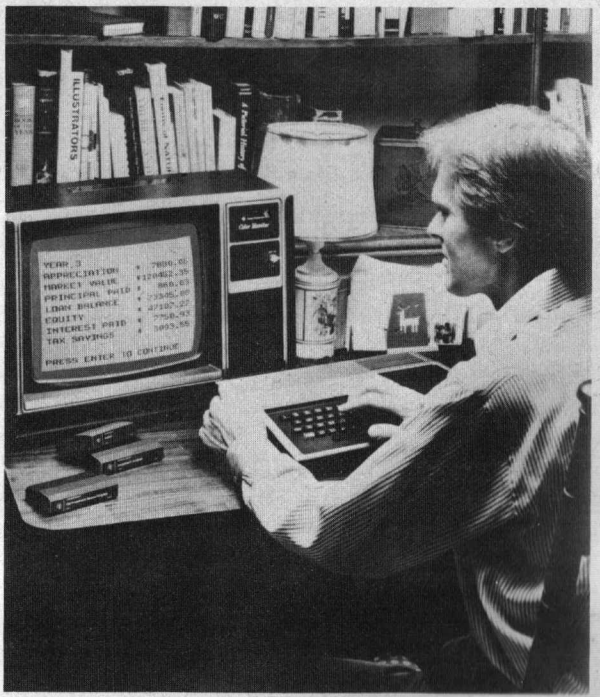


Figure 2-10. The Texas Instruments 99/4 Home Computer (Courtesy of Texas Instruments, Inc., Dallas, Tx)

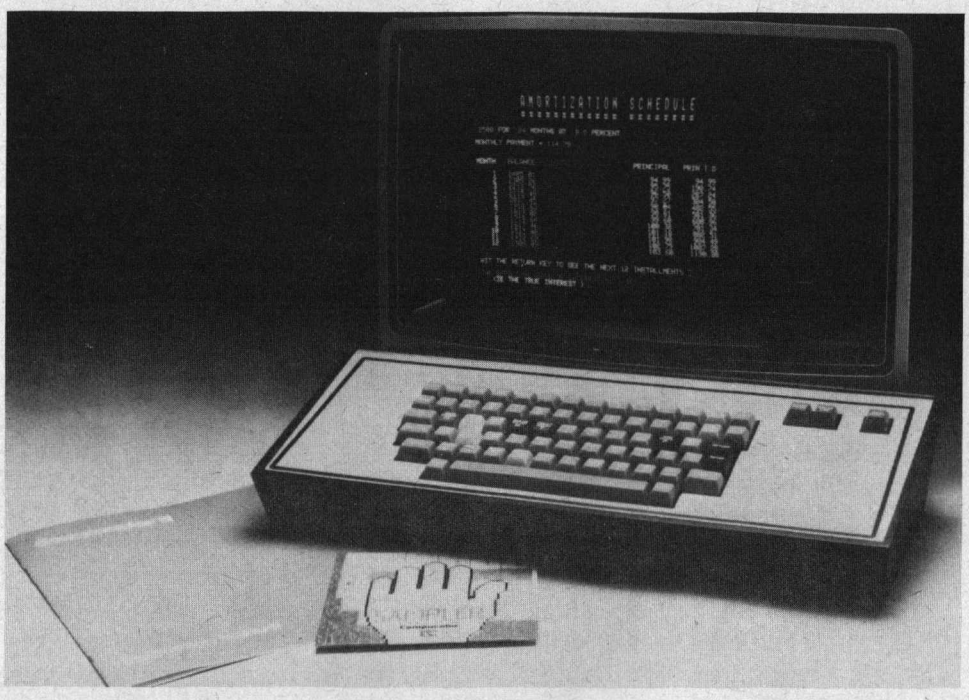


Figure 2-11. The Compucolor II personal computer includes a 13-inch 8-color CRT (Courtesy of Compucolor Corp., Norcross, GA)

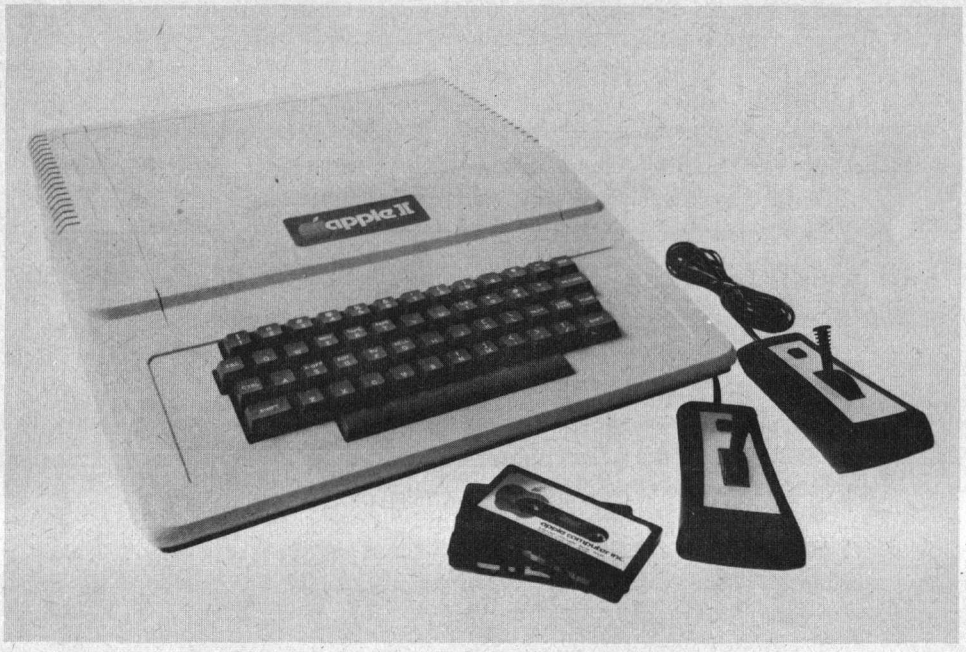


Figure 2-12. The Apple II tabletop computer, an appliance computer that features high resolution color graphics (Courtesy of Apple Computer, Inc., Cupertino, CA)

elaborate programs for systems, business, or specialty uses may cost hundreds of dollars. Appliance computers all have line and figure drawing (graphics) capability, ranging from simple black-and-white graphics (Radio Shack TRS-80) to extensive color graphics (Apple II). All have manuals, supposedly intended for beginners but varying greatly in quality. There are books and magazines (including some supplied on cassette) for the more popular brands, particularly the Radio Shack TRS-80, Apple II, and Commodore PET.

What can you do with the entry-level computer? Typical applications are

- —Games such as chess, checkers, and Star Trek
- —Simple bookkeeping such as checkbook balancing, home budgeting, and small bill paying
- —Trainer for programming classes
- —Drill-and-practice for arithmetic and other subjects
- —Computer-aided-instruction in typing, speed-reading, bridge, and programming
- —Simple business calculations such as loan payments, tax rates, depreciation schedules, and mortgage payments
- —Simple engineering calculations such as those required to determine antenna sizes, measure distances, or determine stresses

Although a video game machine or a programmable calculator could handle some of these applications, a computer has certain advantages:

- —Ability to handle a wider variety of tasks
- —Capacity for more data and results
- —Improved input and output, compared to a calculator, because of the typewriter keyboard and television display
- —Interactive features allowing the computer to ask questions and explain the results
- —Availability of the widely used BASIC computer language
- —Easy expansion for handling larger applications
- —Greater storage capacity for data and programs
- —Wider availability of programs

With some expansion, an appliance computer can perform any data processing tasks, such as

- —Word processing (automatic preparation of letters, invoices, and documents)
- —General business accounting (general ledger, accounts receivable, and accounts payable)
- —Other business tasks (payroll, inventory, mailing lists, and order entry)
- —Data base management and information retrieval
- —Financial planning and simulation
- —Investment tracking and analysis
- —Data analysis
- —Automated design of circuits or mechanisms
- —Legal or medical billing and recordkeeping

These applications are beyond the capabilities of an entry-level computer. In order to perform them, additional equipment may be needed, such as:

—Floppy disks that cost $400 to $1,000 apiece
—A simple printer that costs between $400 and $2,000
—A high-quality printer or typewriter that costs between $1,000 and $4,000
—A terminal (see Chapter 5) that costs between $500 and $2,000
—More memory
—More interfaces
—Containers or cases for the additions, as well as stands, tables, and other furniture
—Programs that may cost as much as $500 each
—Communications interface and programs that allow the computer to talk to another computer or to obtain information over telephone lines
—A larger power supply
—Cables and connectors

The additions may cost more than the original computer. You can expand the entry-level computer gradually but you may end up with a system that is expensive, divided into many separate pieces, cluttered with cords and cables, low in quality, and difficult to use. Appliance computers have been designed for low initial cost and may not provide an adequate foundation for a large system. Too often, manufacturers talk about substantial applications while quoting misleading entry-level prices.

MODULAR COMPUTERS

The next step beyond the appliance computer is the modular or component computer. Modular computers are typically higher in quality and easier to expand than appliance computers but more expensive initially. Leading manufacturers of modular computers include Alpha Micro, Altos Computer Systems, Billings, Cromemco (Figure 1-1), Heath, Ohio Scientific, Pertec, Southwest Technical Products (Figure 2-13), and Vector Graphic (Figure 2-2).

The moudlar computer typically consists of a heavy-duty case with room for several circuit boards. All accessories come separately, just as in a component stereo system. The CPU, each unit of memory, and each peripheral interface occupy one place or "slot." Figures 2-14 and 2-15 show the interiors of modular computers and the slots in which the boards are inserted. Note that you can tell little about a modular computer from the outside (remember the assembled versions in Figures 1-1 and 2-13). All you see are a few switches and lights. The computer's capabilities depend on what is hidden inside. Obviously there is a tremendous temptation for manufacturers and retailers to show the case, list the base price, and describe applications that the computer can only perform if the buyer fills all the internal slots and adds many peripherals.

The basic modular computer costs between $400 and $1,000 and consists of a case, a power supply, and a CPU board. Some entry-level systems also include memory and interfaces for a terminal and a cassette recorder.

Figure 2-13. The Southwest Technical Products Corporation SWTP6800 Personal Computer, a widely used modular computer (Courtesy of Southwest Technical Products Corp., San Antonio, TX)

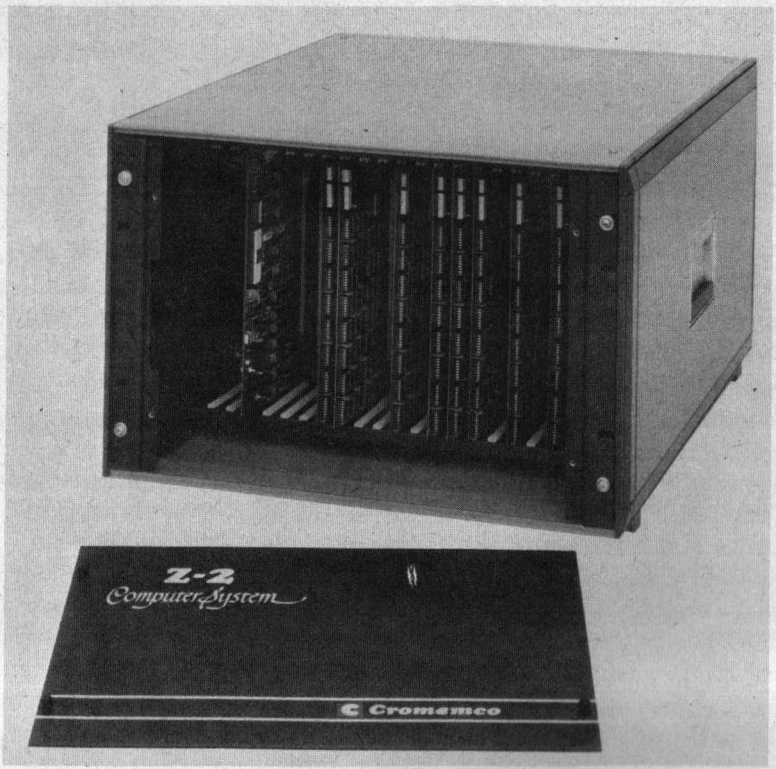


Figure 2-14. An interior view of the Cromemco Z-2 computer, showing the slots into which circuit boards can be inserted (compare to the assembled version of Figure 1-1) (Courtesy of Cromemco, Inc., Mountain View, CA)

You can do almost nothing with this entry-level computer. There is no way to enter data or programs (except with the switches) or to observe the results. There may be no software and almost no memory. Such a computer has little value except as a desk ornament.

To make the computer useful, you must buy

—Additional memory
—An expanded base (or motherboard) that has slots in which other boards can be placed (Figure 2-15 shows a motherboard)
—A terminal or keyboard and video display (and interface)
—An audio cassette recorder and interface

Even with these additions, you will only have an appliance computer in a better case. It will cost more than an appliance computer, typically ranging from $1,000 to $2,000.

You can add more accessories, such as:

—Larger motherboards and extender boards
—Larger power supply
—Disks and the required interfaces
—Printers of various types
—Communications interfaces

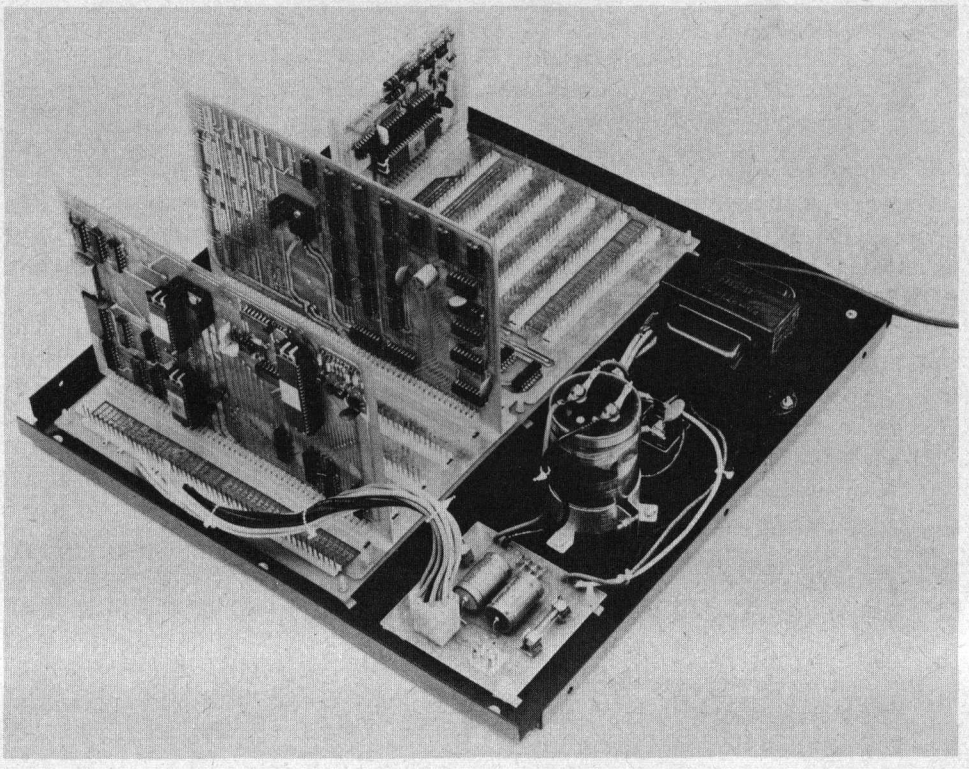


Figure 2-15. An interior view of the Southwest Technical Products SWTP6800 personal computer. The circuit boards are mounted on a base or motherboard. (Courtesy of Southwest Technical Products Corp., San Antonio, TX)

Many manufacturers provide a more substantial initial package at a higher price. A system like the one in Figure 2-16 typically costs between $4,000 and $15,000 and includes

- —CPU
- —Large memory (typically 32K or 64K bytes)
- —Terminal or keyboard and video display
- —Two floppy disks
- —Printer or the interface for one
- —Systems software (including an operating system, editor, debugger, and compilers or interpreters)
- —Business accounting software (for general ledger, accounts receivable, and accounts payable)

This package provides a complete, ready-to-use computer for a small business. The cost obviously depends on how much the manufacturer includes in the package.

Typical applications for modular computers are:

- —Business accounting
- —Word processing
- —Business recordkeeping
- —Data analysis
- —Investment analysis
- —Medical or legal billing and recordkeeping
- —Business forecasting and simulation
- —Information management

The applications are the same as for expanded appliance computers. The difference is in quality and ease of expansion. The situation is much the same as in stereo systems—a low-cost appliance system has the same functions as a higher-cost component system, but the more expensive system performs better, operates more reliably, and is easier to expand or modify.

As for software, the amount available and its quality vary greatly. The minimum software usually is a ROM-based monitor and an assembler, editor, and BASIC interpreter on floppy disk or cassette. Even this software may cost extra, often several hundred dollars more.

Other software that may be available includes

- —Disk-based operating systems
- —More powerful BASIC interpreters
- —Other computer languages (COBOL, FORTRAN, PASCAL, PILOT, APL, and C)
- —Word processing systems
- —Accounting or bookkeeping programs
- —More powerful editors
- —Debuggers
- —Communications programs
- —Graphics or plotting programs
- —Engineering programs
- —Simulation programs
- —Specialty programs for doctors, dentists, lawyers, realtors, teachers, distributors, automobile dealers, accountants, wholesalers, and others

Figure 2-16. The Cromemco System Three Disk Model Computer, a packaged modular computer that includes several floppy disks and software packages (Courtesy of Cromemco, Inc., Mountain View, CA)

There is some independently produced software for the most popular computers. But the market for such software is much smaller and the cost is correspondingly higher than for appliance computers. As we mentioned, the cost of developing a program does not depend on the number of copies sold. Thus programs for more popular computers can be sold at lower prices, since the original cost can be recouped by selling more copies.

Programs for modular computers typically cost $50 to $1,000, and one can easily spend several thousand dollars on a few large programs. A rule of thumb in serious computing is that one should expect to spend at least as much on software as on hardware.

Among the leading independent developers of hardware and software for modular computers are

—Digital Research of Pacific Grove, CA, developers of the CP/M (Control Program/Microcomputer) operating system and its associated programs.

—Microsoft of Bellevue, WA, developers of BASIC, FORTRAN, COBOL, and PASCAL compilers and interpreters for personal com-

puters. Microsoft also produces assemblers, editors, and other programs that are included with many computers.

—Michael Shrayer Software of Palm Springs, CA, developers of the Electric Pencil Word Processing System. This is a text processing program that formats and justifies letters or documents and performs such tasks as hyphenation, numbering and titling of pages, scrolling (moving text up and down on the screen), line and paragraph indentation, centering, and underlining.

—Tarbell Electronics of Carson, CA, developers of cassette and disk interfaces.

—Godbout Electronics of Oakland, CA, developers of memory boards and accessories (such as motherboards).

—Thinker Toys of Berkeley, CA, developers of memory boards and interfaces (products of Morrow's Micro-Stuff).

—Softech Microsystems of San Diego, CA, suppliers of UCSD PASCAL. This popular software package, originally developed at the University of California, San Diego, includes an operating system, compilers, editors, assemblers, and other systems programs. UCSD PASCAL was designed specifically to provide a complete set of systems programs that could run on almost any computer and would be easy to use.

We mention these names because of frequent references to them in books, magazine articles, and advertisements. We are not endorsing their products; others produce similar products that may be of equal or higher quality.

Most modular computers use an interconnection (*bus*) standard. The most popular standard is the S-100 bus, consisting of 100 parallel electrical connections. Other buses include SS-50, Heath, and Ohio Scientific Instruments. Many manufacturers produce accessories for S-100 bus computers, whereas only a few deal in products for the other buses. Chapter 6 treats buses and interfacing in more detail.

CENTRAL PROCESSING UNIT

We will now describe individual components at greater length, starting with the CPU and proceeding through the memory and the essential programs. As we mentioned previously, the central processing unit is the "brain" of the computer. It directs operations, executes programs, moves data from one place to another, and performs arithmetic and other tasks.

The CPU normally consists of one device, a *microprocessor.* Several different microprocessors are widely used in personal computers; their capabilities are similar although each processor has its own advocates. We do not describe these devices in detail but we discuss their position in the marketplace and how that affects users. The common processors are

—Intel 8080
—Motorola 6800
—Zilog Z80
—MOS Technology 6502

The Intel 8080 is an early microprocessor (dating from the ancient days of 1973) and is currently the most popular for industrial applications. The S-100 bus was originally developed for the 8080 processor as was such software as the CP/M operating system and the Electric Pencil Word Processing System. Manufacturers of 8080-based computers include Billings, Compucolor, Heath, North Star, Northwest Microcomputer, Pertec, Space Byte, and Vector Graphic.

The Zilog Z80 is a close relative of the Intel 8080; it is faster and more powerful than the 8080 but can run most of the same programs. Z80 CPU boards are commonly available for the S-100 bus and most Z80-based computers can run the CP/M operating system and the Electric Pencil. Manufacturers of Z80-based computers include Altos Computer Systems, Billings, Cromemco, Durango Systems, Exidy, Radio Shack, and Vector Graphic.

The Motorola 6800 is very different from the 8080 or Z80. The 6800-based computers do not generally use the S-100 bus nor can they run CP/M or the Electric Pencil. Buses and operating systems are available for 6800-based computers, but they are far less popular than S-100 and CP/M. Manufacturers of 6800-based computers include Gimix, Midwest Scientific Instruments, Smoke Signal Broadcasting, and Southwest Technical Products.

The MOS Technology 6502, a relative of the Motorola 6800, is widely used in one-board and appliance computers. The 6502-based computers generally do not use any of the standard buses nor can they run popular software such as CP/M or the Electric Pencil. However, many large manufacturers of 6502-based computers provide extensive lines of hardware and software; these manufacturers include Apple, Atari, Commodore, Ohio Scientific Instruments, Rockwell, and Synertek.

New microprocessors are always appearing. Some are more powerful than those currently in use and will ultimately replace them. Such devices include the Intel 8086, Motorola 6809 and 68000, and the Zilog Z8000.

Most users do not care about what microprocessor is inside their computers, any more than most drivers care about what model of engine or transmission is inside their automobiles. If you buy an appliance computer or a complete packaged system, use only the accessories and software intended specifically for it, and write programs in a high-level language like BASIC or PASCAL, then you need not even be aware of what microprocessor is inside.

The microprocessor is of concern, however, to those who

- —Have a one-board computer rather than an appliance computer
- —Want to construct their own systems or add special software or peripherals
- —Write programs in assembly language rather than in high-level languages
- —Use computers to control equipment or perform other tasks that require detailed understanding of the computer and precise timing
- —Want to write or use special systems software

The typical business, scientific, or classroom user is no more concerned with the microprocessor than with the power supply, fan, or case.

What, then, does the microprocessor determine? It determines

—How the computer is programmed at the assembly language level
—What programs will run on the computer. As we have noted, more software (such as the CP/M operating system and the Electrical Pencil word processing system) is available for 8080 or Z80-based computers than for any others. This is by far the most important difference for most users.
—What accessories can be attached to the computer. Here again, the most widely available accessories are those for the S-100 bus, which is almost always based on an 8080 or Z80 microprocessor.

RAM

We have already distinguished ordinary memory (RAM) from permanent memory (ROM). The user is far more likely to buy additional RAM than ROM, although both are available for most computers. RAM boards such as the one in Figure 2-17 are always popular. The beginner soon learns that a computer never has enough memory. As with file cabinets or closets, things magically appear to fill and then overflow the available memory. Fortunately, memory is going down in price. In fact, from 1975 to 1978 alone, memory costs dropped by a factor of four despite the general inflation of that period.

Typical characteristics of RAM boards include the amount of memory, computer or bus compatibility, power consumption, and speed. High-volume memory boards (such as 16K boards for the S-100 bus) are far less expensive on a per-byte basis than are boards in less popular sizes or for less popular computers.

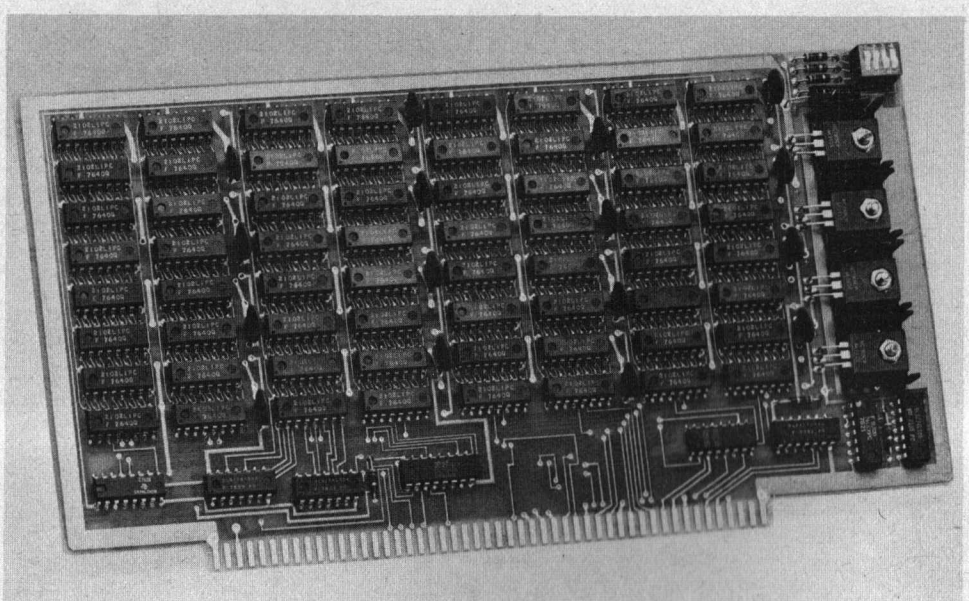


Figure 2-17. The Vector Graphic 8K RAM Board (Courtesy of Vector Graphic, Inc., Westlake Village, CA)

ROM

ROM boards are not nearly as popular as RAM boards. You may want a ROM board for a monitor or operating system; permanent versions of constantly used programs such as assemblers, compilers, or editors; final versions of controller programs; lookup tables; and peripheral drivers.

Typical features of ROM boards (Figure 2-18) are size, computer or bus compatibility, and power consumption. Sometimes ROMs come with programs already in them. Otherwise the user must place a program or data in the ROM; this is referred to as *programming* or *burning* the ROM and requires special equipment. Burning a ROM is like carving a stone tablet. Some suppliers will program ROMs for users; some boards, like the Cromemco Bytesaver in Figure 2-18, include programming capability.

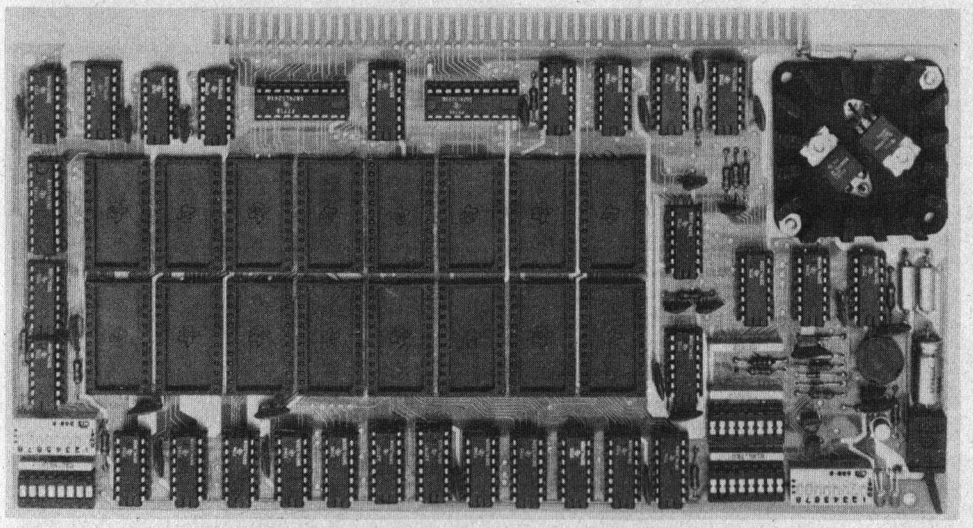


Figure 2-18. The Cromemco Bytesaver PROM Board with on-board programming capability (Courtesy of Cromemco, Inc., Mountain View, CA)

OPERATING SYSTEMS

The operating system is the computer's chief executive. The exact functions of operating systems (like those of chief executives) vary greatly, but typical ones include

- —Starting the computer properly, often called a *system generation* or *sysgen*
- —Allowing the user to run system and user programs
- —Allowing the user to save data and programs and reload them later; information is saved in a *file* that can be named and then stored and retrieved by name
- —Handling minimum input/output; this may mean a keyboard and display for a one-board computer, or a terminal and floppy disks for a packaged disk-based system

—Providing debugging aids so the user can stop and start programs, examine and change the contents of memory, and trace the execution of programs
—Providing error recovery, error messages, and system supervision
—Keeping track of elapsed time

The most popular types of operating systems are

1. *A simple ROM-based monitor:* this handles basic I/O, data loading, and debugging. The computer and the user are helpless without such a monitor; before ROMs were commonly available, users often had to load a startup program (or *bootstrap*) by hand from front panel switches. A ROM-based monitor is necessary to get the computer started even if you have a more elaborate operating system.
2. *A cassette operating system:* this transfers programs and data to or from cassettes. Most appliance computers have a ROM-based cassette operating system that is closely linked with their BASIC interpreters.
3. *A disk operating system or DOS:* this transfers data and programs to or from disks. Disk operating systems are normally far more complex than cassette operating systems and have far greater capabilities. As we have noted, the CP/M operating system from Digital Research has become a standard for personal computers.

ASSEMBLERS

Assemblers translate the simple human-readable form of a computer program (an assembly language program or *source code*) into the numbers that a computer can execute (a machine language program or *object code*). Figure 2-19 shows an example of the use of an assembler; the source code is on the right and the object code on the left. Assembly language is referred to as *low-level* or *machine-oriented language* since it is specific to a particular CPU (microprocessor) and is translated line-for-line into computer instructions.

Most programmers only use assembly language when no high-level language is available that can provide a convenient way to express what the computer is supposed to do. One-board computers like the MOS Technology KIM usually have no high-level language at all. Applications for which no high-level language is convenient typically involve control of equipment, detailed input and output, or precise timing rather than games, computer-aided instruction, or business data processing.

The trend in computers is clearly toward greater use of high-level languages. New features are often added to languages like BASIC or FORTRAN to handle additional applications. But most programmers still find a knowledge of assembly language to be useful on occasion, particularly to write input and output routines, use programs that are part of the operating system (such as the programs that read the keyboard or write on the video display in an appliance computer), and understand the detailed workings of programs. Knowing assembly language for a particular computer is much like knowing how to fix your car; you can drive the car without knowing any auto mechanics, but a little knowledge is extremely handy once in a while.

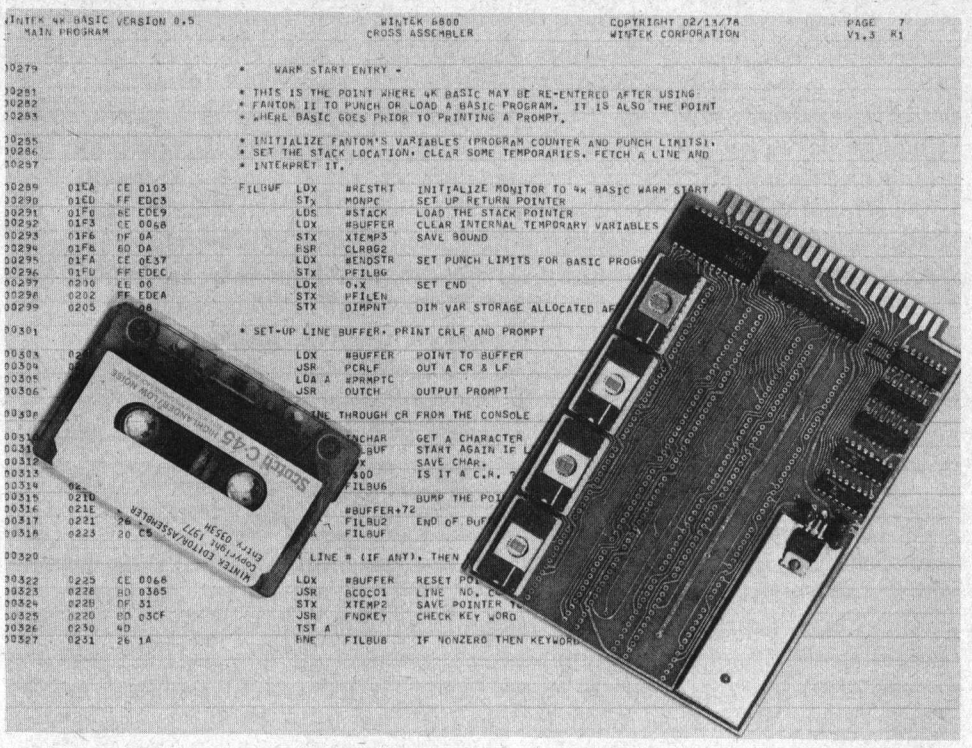


Figure 2-19. A typical example of an assembled program, including both source code (on the right) and object code (on the left). The assembler can be obtained either as a cassette or in PROMs on a ROM board. (Courtesy of Wintek Corp., Lafayette, IN)

COMPILERS AND INTERPRETERS

Compilers and interpreters translate programs written in high-level languages into actual computer instructions. A *compiler* translates the entire program and produces a translated version (called an *object program*) for later use, whereas an *interpreter* translates and executes the program line by line without producing an object program. Compilation is obviously better for programs that you plan to run repeatedly (since they only have to be translated once). Interpreting is better for programs that you are still developing, since you learn about errors immediately. Most translators for BASIC are interpreters, whereas most translators for COBOL, FORTRAN, and PASCAL are compilers.

Most high-level languages conform to standards, although there is no penalty for deviation. BASIC is the least standardized language; there are numerous versions, much like dialects of a human language. Although all the common languages can be used in a variety of applications, each has a special orientation that depends on its original date of development, its purpose, and its audience. FORTRAN (*For*mula *Tran*slation Language) and BASIC were originally intended for engineers and scientists who want to solve numerical problems involving formulas and equations. COBOL (*C*ommon *B*usiness *O*riented *L*anguage) was originally intended for data processing

applications involving complex records (such as payroll files that include name, address, job classification, date of hire, wage rate, tax withholding rates, etc.) and large amounts of output in a precise format (such as paychecks, utility bills, or bank statements). PASCAL is a more recent language developed by computer scientists with the aim of promoting generality and clear and efficient program writing.

We should note that programs written in high-level languages usually run slower and occupy more memory than those written in assembly language. BASIC programs are particularly slow because they must be translated each time they are run. But even compiled programs are seldom as fast or as compact as well-written assembly language programs. The reason is that the translator programs work mechanically and cannot match a skilled assembly language programmer, just as an automatic transmission cannot match the gasoline mileage of a manual transmission (with a skilled driver). Some programmers, therefore, use assembly language to speed up or shorten sections of a program that is written in a high-level language.

On the other hand, high-level languages are easier to use and learn (the same as an automatic transmission), require less skill and detailed knowledge, and are more independent of particular computers than are machine or assembly languages. Furthermore, there are many books, manuals, and articles describing the use of common high-level languages and a large backlog of programs. The programs themselves are easier to understand, use, and modify than are assembly or machine language programs.

High-level languages do have some disadvantages besides producing programs that require extra time and memory. Like an automatic transmission, they make precise control difficult; high-level languages limit the programmer's access to the computer, restrict input and output, and interfere with the timing of operations. Most programs, however, do not require precise control, any more than normal highway driving does.

Although a high-level language is independent of a particular computer, the compiler or interpreter is not. Compilers or interpreters will only run on computers that have a particular CPU, set of peripherals, operating system, and other software.

EDITORS

We have already noted some of the features and uses of editors. These programs are normally used to prepare programs for compilation or assembly; prepare data or text; save data, text, or system commands that will be used repeatedly, thus saving repetitive typing of the same material; make corrections in programs, data, or text; and revise, analyze, or rearrange text. There are various kinds of editors, including character editors that handle one character at a time, line editors that handle one line at a time, word processors, and screen editors that use the screen of a CRT like a printed form.

Typical tasks that an editor can perform are:

—*Insertion*, so you can add extra text or correct an omission
—*Deletion*, so you can remove unwanted text, stray marks, or extra characters

—*Concatenation,* so you can combine programs or data and text. (For example, you might want to add a standard warranty and service agreement section to the end of several manuals.)
—*Search,* so you can locate a particular line, character, or phrase
—*Move,* so you can reorder or reorganize text
—*Replace,* so you can correct errors or change text (For example, you might want to replace every occurrence of "JOHN SMITH" with "JON SMITH" to correct a spelling error or replace "110V" with "220V" to reword a manual for European use.)
—*Merge,* so you can combine files in a particular order
—*Save,* so you can retain a file for later use. A file is just a related set of information, much like the materials in a file cabinet.

DEBUGGERS

A good debugger is vital if you are going to write programs. Without one, identifying, locating, and correcting errors becomes extremely tedious. We describe the purpose and use of debuggers in more detail in Chapter 4. The debugger, like the editor, is often part of the operating system and closely linked to an assembler, a compiler, or an interpreter.

Typical features of a debugger are

—*Breakpoints* that can stop the program under specified conditions
—*Traces* that describe the operation of the program in detail
—*Single-step mode* that executes a program one statement at a time
—*Dumps* that print the contents of many memory locations at one time
—*Exercisers* that test branches and logic in a program
—*Input/output generators* that allow you to simulate or control input and output devices

SUMMARY

In this chapter we have discussed what a useful computer must include and what options are commonly available. A useful computer consists of compatible hardware and software. The essential hardware is a CPU, memory (permanent—ROM—and temporary—RAM), input and output peripherals, mass storage devices and interfaces. The essential software is an operating system, other systems software, and applications programs. The performance of many tasks also requires a well-organized collection of data or data base.

There are three types of personal computers.

—*One-board computers,* aimed at the engineer or electronics hobbyist
—*Home or appliance computers,* low-cost entry-level systems for data processing
—*Modular computers,* more power and higher quality than the appliance computers at a higher initial cost

One-board computers are intended for simple engineering applications involving only low-cost peripherals and little software. Appliance and modular computers can handle many applications, including business recordkeeping, word processing, computer-aided instruction, and scientific calculations. They differ in quality and ease of expansion.

Typically the essential hardware is implemented as follows in personal computers:

—The *CPU* is a one-chip microprocessor. The particular device involved does not matter to most users, but it does affect the accessories and programs that can be utilized with the computer.

—*Memory* consists of RAM and ROM. RAM is the more customary changeable memory but its contents are lost when the computer is turned off. ROM is permanent and cannot be changed in normal operation; its availability has made computers easier to use and has made the ready-to-go entry-level computer possible.

The essential software can be implemented in many ways but typically includes

—An *operating system* that acts as an overall administrator

—*Assemblers* and *compilers* that translate programs written in human-readable form into the numerical codes that the computer can execute

—An *editor* that allows the user to prepare data and printed matter

—A *debugger* that helps the user find and correct errors in programs

Most appliance computers combine these programs in a single ROM-based package. When purchasing programs (or accessories), a user must determine whether they will work with his or her computer; programs and accessories generally require a particular computer and a particular set of peripherals and other programs.

3 Introduction to BASIC Programming

The most convenient way to program a computer is to use a high-level language. BASIC, first developed at Dartmouth College in the middle 1960s, is the most popular language for personal computers because it is easy to learn, use, and implement. This chapter describes how to write simple programs in BASIC. We emphasize BASIC because of its widespread use, but we stress fundamental approaches to programming that do not depend on which high-level language you use. The references at the end of this chapter describe BASIC in more detail than we can provide here. For descriptions of other languages, we suggest that you consult the popular computer magazines such as *Byte* and *Microcomputing* (see Chapter 8 for their addresses).

In this chapter, we use only the elementary features of BASIC. The example programs can therefore run on most personal computers that have BASIC. We have made the programs simple, general, and easy to understand, rather than particularly efficient. You can probably improve our programs by using features specific to your computer's version of BASIC.

PROBLEM EXAMPLES

We describe BASIC in terms of three examples.

—Example 1: An arithmetic quiz program that tests students on sums.

—Example 2: An inventory control program that keeps track of the number of items in stock.

—Example 3: A Monopoly program that simulates the playing of the popular board game.*

None of these problems involves any mathematics beyond arithmetic. We concentrate on writing programs that perform routine tasks in a way that shows the value and power of computers.

*Monopoly is a trademark of the Parker Brothers Division of General Mills, Inc.

The programs for these examples must

- —Read the data.
- —Perform operations in a specified order.
- —Print the results.
- —Perform various procedures depending on the values of data or results.
- —Repeat operations.
- —Ask the operator questions and respond to the answers.
- —Handle sets of data such as purchases or sales for a whole group of items or customers.
- —Analyze the data, e.g., find the average value, arrange the data in numerical order, find the largest or smallest items, or determine if an item falls in a category.
- —Handle data that is not numerical, like answers that can only be yes or no, the ownership of a property, the eligibility of a player to move, or the state of development of a property (for example, whether it is mortgaged and whether houses or a hotel can be or have been built on it).

We begin by using the computer as a calculator; we provide it with data, then make it perform some simple arithmetic and report the result. This type of operation is necessary to calculate sums or differences, determine inventory levels from sales and purchases, and increase or decrease the amount of money that Monopoly players hold. We then add *loops* and *decisions,* thus enabling the computer to repeat procedures and to decide whether an answer is correct, an inventory level is too low or too high, or a player has passed GO (and is entitled to collect $200 and start around the board again). This capacity to repeat procedures and make decisions transforms the computer from a calculator into an independent agent able to work continuously and intelligently without apparent direction. We then make the programs interactive so that the user can ask the computer to continue or terminate a quiz, print selected results, or perform a property transaction. Interactive programs give the computer human qualities, since it can ask questions, offer comments or help, and respond to answers or requests. Finally we introduce the ability to handle sets of data, so that the computer can keep track of many students, inventory items, or game players at the same time.

INTRODUCTION TO BASIC

The primary features of BASIC are as follows.

- —Each line of a BASIC program must have a number. The line numbers can have any values, but programmers usually space them at least ten apart (e.g., 10, 20, 30) so they can insert lines in-between later.
- —The computer executes the individual instructions or *statements* (when told to do so) in numerical order by line number, e.g., 20 after 10, 30 after 20.

You can enter lines in any order. The computer arranges the statements in numerical order automatically, thus allowing you to insert lines at any point as long as you have left gaps in the line numbers.

Besides statements, BASIC also has commands that do not require line numbers. The most common commands are

NEW or SCR (SCRATCH): clear the computer, i.e., start with a clean slate.

LIST: list the program; i.e., print all statements in the order determined by increasing line numbers.

RUN: execute the program, starting at the statement with the lowest line number.

Once your computer is working in BASIC (this may require some preliminary steps), the procedure for writing and executing a program is

—Type NEW or SCR to clear the computer. This command erases old programs.
—Enter numbered lines in any order. We will put only one statement on a line, although most versions of BASIC allow more.
—Type LIST to produce an ordered listing of the program that you can check.
—Type RUN to execute the program. Remember, the computer will not produce any results until it actually executes the program.

You must conclude each entry or command with a (CARRIAGE) RETURN. Many computer keyboards have a special key marked ENTER or RETURN for this purpose.

If you make a typing error or some other mistake, you can rectify it by entering the correct line with the old line number. BASIC erases the old, incorrect line automatically. If you want to delete a line, just type its line number followed by RETURN or ENTER. Of course you can also use your editor to make corrections or changes.

INITIAL PROGRAMS

The first BASIC statements that we discuss are

```
LET (assign a value)
PRINT (print the results)
END (end of program)
```

A LET statement assigns the value on the right-hand side of an equation to the variable on the left-hand side. Variables have names consisting of a letter or a letter followed by a digit. The right-hand side may also contain arithmetic operations such as

+: addition
−: subtraction
*: multiplication (watch this one)
/: division

Typical LET statements are

```
10 LET A = 3
20 LET R7 = S4 + T3
30 LET A = H * L
```

The *left-hand side* of a LET statement always consists of a single variable; the *right-hand side* may contain variables, operations, and numbers. BASIC uses * (asterisk) rather than × for multiplication and / (slash) rather than ÷ for division. The multiplication sign cannot be omitted.

Each variable occupies a place in the computer's memory, so

```
10 LET A = B + 1
```

means "take the value in memory location B, add 1 to it, and place the result in memory location A." Of course, the memory locations really have numerical addresses but the computer itself keeps track of which numerical addresses correspond to which variables. For example, A might be memory location 2000 and B memory location 3500.

A PRINT statement prints the current value of a variable or an expression. Typical PRINT statements are

```
 10 PRINT X
 50 PRINT A, B, C + D
100 PRINT "ANSWER IS", A
200 PRINT 7 * 5
```

You must separate entries with commas (or semicolons on some computers). You can add text (typed characters) such as headings, questions, names or descriptions, units, messages, dollar signs, or punctuation marks by placing the characters in quotation marks. The computer will print the text along with the numerical values.

An END statement marks the end of the BASIC program. We usually assign a high line number such as 999 to the END statement, in order to keep it out of our way. Every program must have an END statement.

We can now write simple programs for our three examples.

Example 1—arithmetic quiz: Add the numbers 5 and 8.

```
 10 LET A = 5
 20 LET B = 8
 30 LET C = A + B
 40 PRINT C
999 END
```

We have implemented each assignment of a value or calculation as a LET statement. You can enter the statements in any order. When you type RUN, the computer will execute the statements in numerical order by line number, that is, first line 10, then 20, 30, 40, and finally 999.

We can make the output easier to understand by printing some text along with the numerical values.

```
 10 LET A = 5
 20 LET B = 8
 30 LET C = A + B
 40 PRINT "THE SUM OF", A, "AND" B, " = ", C
999 END
```

Note that we must place the equal sign and the spaces inside the quotation marks with the letters.

Example 2—inventory control: Calculate current inventory level (C) from former inventory level (25) and sales (10).

```
 10 LET F = 25
 20 LET S = 10
 30 LET C = F - S
 40 PRINT "OLD LEVEL   SALES   NEW LEVEL"
 50 PRINT F, S, C
999 END
```

The computer prints the text (including spaces and punctuation) exactly the way you type it inside the quotation marks. If you want to move the headings or align them with data from later PRINT statements, you can simply change the spacing in the statement that prints the headings. Remember that you can insert spaces (like any other character) anywhere inside the quotation marks—at the beginning, between titles, or between words in a title. Either counting the number of spaces you need or simple trial-and-error will produce reasonable printed output for many programs. There are more precise ways to create forms on most computers (much like using the tabs and other features on typewriters), but we will not discuss them here. If you want the computer to skip a line (for example, between the headings and the data), you can use a PRINT statement without any text or variables. For example, adding the statement

```
45 PRINT
```

to the last program will result in a blank line between the titles and the data.

We can add other data to the inventory problem, such as purchases (10), returns (3), and losses (2).

```
 10 LET F = 25
 20 LET S = 10
 22 LET P = 10
 25 LET R = 3
```

```
 28 LET L = 2
 30 LET C = F - S + P + R - L
 40 PRINT C, F, S, P, R, L
999 END
```

We must be careful not to use the same name twice by accident. For example, we might want to use L for both losses and inventory level. But the location L in the computer's memory can hold only one value at a time. New versions of BASIC make duplication less likely by allowing longer names. Longer names also make programs easier to understand; for example, a statement like NEWLEV = OLDLEV − SALES is more meaningful than our C = F − S.

Example 3—Monopoly: Add $200 to a player's cash holding ($2,500).

```
 10 LET C = 2500
 20 LET B =  200
 30 LET C = C + B
 40 PRINT "NEW CASH HOLDING = $",  C
999 END
```

The variable on the left-hand side of a LET statement can also appear on the right-hand side, thus allowing us to update a holding or the position of a piece on a game board. The statement LET C = C + B means "place in memory location C the old value from memory location C plus the value from memory location B." Our simple program only handles numbers; it does not handle dollar signs, commas, colons, or other symbols. We must write LET B = 200, not I ET B = $200. However, we can include punctuation or symbols in the results by placing the required characters in quotation marks.

BRIEF REVIEW 1

So far we have introduced the following commands and statements.

COMMANDS

LIST
NEW or SCR (SCRATCH)
RUN

STATEMENTS

END
LET
PRINT

Remember the following guidelines.

—Each line of a BASIC program must have a number. The computer executes the instructions (or *statements*) in numerical order by line number.

—Variable names consist of a letter or a letter followed by a digit. These names correspond to locations in the computer's memory.
—Each calculation or assignment of a value requires a LET statement. The left-hand side of the LET statement is the name of a variable and the right-hand side is an expression consisting of variables, numbers, and arithmetic operations. The computer places the value of the expression in the memory location that holds the variable.
—BASIC uses an asterisk (*) for multiplication and a slash (/) for division. You cannot omit the multiplication sign; the computer treats AB as an error, not as A times B.

REPEATING OPERATIONS

Obviously we don't need a computer to add or subtract two numbers. We can do that by hand or on a calculator. The computer becomes useful when the amount of data is large.

The BASIC statements that we need in order to handle more data are

DATA (hold a list of values)

READ (assign values from a DATA statement)

DATA statements contain lists of values. The computer uses the values in the order in which they are typed on the line. If you run out of space on the line, you must place the remaining values in another DATA statement with a different line number.

READ statements tell the computer to get values for variables from DATA statements. The computer assigns values to all the specified variables, starting with the lowest numbered DATA statement and proceeding left-to-right through as many DATA statements as it needs (in numerical order). It uses each value once. If it runs out of values, the computer terminates the program and prints an error message such as "OUT OF DATA."

A typical combination of a DATA statement and a READ statement is

```
10 DATA 1, 10, 20
20 READ A, B, C
```

When the computer executes line 20, it assigns the first value in the DATA statement (1) to the first variable (A), the second value (10) to the second variable (B), and the third value (20) to the third variable (C). Thus executing the READ statement has the same effects as executing the following three LET statements

```
10 LET A = 1
20 LET B = 10
30 LET C = 20
```

The combination of a DATA statement and a READ statement is shorter than a series of LET statements when there are more than one or two variables involved. The combination also has other advantages which we will see later when we write programs that execute READ statements repeatedly.

Another example of the combination is

```
10 DATA 5
20 DATA 10
30 READ M, N
```

When the computer executes line 30, it assigns the value in the first DATA statement (5) to the first variable (M) and the value in the second DATA statement (10) to the second variable (N). The computer does not actually execute DATA statements; it merely records the values in them for use by READ statements.

When the computer executes READ statements, it assigns the values automatically. You must line up the values in the DATA statements with the variables in the READ statements; the computer will not catch any mistakes. A program can contain any number of DATA statements and they may come anywhere in the program; DATA statements may be located before or after the READ statements that use them. The computer will terminate the program immediately if it is asked to assign a value when it has none left; it will not assume that the missing value is zero.

Another important BASIC statement is

```
GOTO (execute the specified line number next)
```

GOTO statements tell the computer to execute the specified line number next rather than continuing its normal sequence. Thus

```
20 GOTO 80
```

means "execute line 80 next." The computer then proceeds normally, starting at line 80.

A GOTO statement only tells the computer which line to execute next; no variables change value. Since the computer executes lines in numerical order automatically, you only need a GOTO statement if you want the computer to skip ahead or move backward. A GOTO statement acts like an arrow directing traffic in something other than its normal (straight ahead) direction.

We can now expand our simple programs to handle more data.

Example 1—arithmetic quiz: Add numbers and print the sums.

```
 10 DATA 5, 8, 2, 3, 4, 1, 7
 20 READ A, B
 30 LET C = A + B
 40 PRINT "THE SUM OF", A, "AND", B, " = ", C
 50 GOTO 20
999 END
```

Line 50 (GOTO 20) sends the computer back to line 20 so it does the next sum; the results printed are

```
THE SUM OF 5 AND 8 = 13
THE SUM OF 2 AND 6 = 8
THE SUM OF 3 AND 4 = 7
THE SUM OF 1 AND 7 = 8
OUT OF DATA IN LINE 20
```

The exact form of the printed results and the error message vary from computer to computer. Many computers leave a large number of blank spaces ahead of the actual values unless you replace the commas in the PRINT statement with semicolons.

Example 2—inventory control: Calculate current inventory levels for many items from former levels, sales, and purchases.

```
10 PRINT "OLD LEVEL"        SALES        PURCHASES        NEW LEVEL"
20 DATA 20, 5, 10, 40, 10, 0, 30, 5, 5
30 DATA 50, 1, 8, 30, 7, 2, 40, 12, 10
40 READ F, S, P
50 LET C = F - S + P
60 PRINT F, S, P, C
70 GOTO 40
999 END
```

The computer only prints the headings once since it only executes line 10 once. You can adjust the spacing on line 10 to align the headings with the columns of numbers; remember that you can insert extra spaces anywhere inside the quotation marks. As in the previous example, the computer quits when it runs out of values.

The results are

```
OLD LEVEL        SALES        PURCHASES        NEW LEVEL
    20             5              10              25
    40            10               0              30
    30             5               5              30
    50             1               8              57
    30             7               2              25
    40            12              10              38

OUT OF DATA IN LINE 40
```

Example 3—Monopoly: Add amounts to all players' cash holdings.

```
10 PRINT "NEW CASH HOLDINGS"
20 DATA 2500, 200, 3000, 100, 2720, 150, 2000, 50
30 READ C, B
40 LET C = C + B
50 PRINT C, "DOLLARS"
60 GOTO 30
999 END
```

The printed results are

```
NEW CASH HOLDINGS
2700 DOLLARS
3100 DOLLARS
2870 DOLLARS
2050 DOLLARS
OUT OF DATA IN LINE 30
```

BRIEF REVIEW 2

We have introduced the following statements.

```
DATA
READ
GOTO
```

Remember these guidelines.

—Each line, even if it only contains a DATA statement, must have a number.
— A READ statement causes the computer to assign values to variables from DATA statements. The computer uses each value once and assigns values in the order specified in the READ statement.
—The computer stops the program and prints an error message when it runs out of values.
—A GOTO statement tells the computer to execute a specific line number next. You can use a GOTO to send the computer back to a previous statement or to have it skip statements.

COUNTING

Simple counting is necessary in many programs. For example, we might want the computer to count the number of problems and the number of correct answers in the arithmetic quiz, the number of items in the inventory problem, or the number of players and the number of rounds in the Monopoly game.

The computer can count just as a person does. The idea is to add 1 to the count for each new item or event. If I is the count, the basic operation is

```
LET I = I + 1
```

However, we must also explicitly start I at zero.

```
LET I = 0
```

You cannot assume that a variable is zero just because you haven't used it. Note that you must organize the program so that the computer only executes LET I = 0 once initially, while it executes LET I = I + 1 each time it is supposed to count.

We can, for example, count repetitions in our earlier programs as follows.

Example 1—arithmetic quiz: add numbers and print the sums.

```
 10 DATA 5, 8, 2, 6, 3, 4, 1, 7
 20 LET I = 0
 30 READ A, B
 40 LET C = A + B
 50 LET I = I + 1
 60 PRINT I, "THE SUM OF", A, "AND", B, " = ", C
 70 GOTO 30
999 END
```

The computer executes line 20 only once, whereas it executes line 50 four times. The results are

```
1 THE SUM OF 5 AND 8 = 13
2 THE SUM OF 2 AND 6 = 8
3 THE SUM OF 3 AND 4 = 7
4 THE SUM OF 1 AND 7 = 8
OUT OF DATA IN LINE 30
```

Example 2—inventory control: Calculate current inventory levels for many items from former levels, sales, and purchases.

```
 10 PRINT    "ITEM NO    OLD LEVEL    SALES    PURCHASES    NEW LEVEL"
 20 DATA 20, 5, 10, 40, 10, 0, 30, 5, 5
 30 DATA 50, 1, 8, 30, 7, 2, 40, 12, 10
 40 LET I = 0
 50 READ F, S, P
 60 LET C = F - S + P
 70 LET I = I + 1
 80 PRINT I, F, S, P, C
 90 GOTO 50
999 END
```

The results are

ITEM NO	OLD LEVEL	SALES	PURCHASES	NEW LEVEL
1	20	5	10	25
2	40	10	0	30
3	30	5	5	30
4	50	1	8	57
5	30	7	2	25
6	40	12	10	38

OUT OF DATA IN LINE 50

Example 3—Monopoly: Add amounts to all players' cash holdings.

```
 10 PRINT              "PLAYER NO.    NEW CASH HOLDINGS"
 20 DATA 2500, 200, 3000, 100, 2720, 150, 2000, 50
 30 LET I = 0
 40 READ C, B
 50 LET C = C + B
 60 LET I = I + 1
 70 PRINT I, C, "DOLLARS"
 80 GOTO 40
999 END
```

The results are

PLAYER NO.	NEW CASH HOLDINGS
1	2700 DOLLARS
2	3100 DOLLARS
3	2870 DOLLARS
4	2050 DOLLARS

OUT OF DATA IN LINE 40.

MAKING DECISIONS

So far, we have used the computer much like a phonograph with an automatic turntable. The computer performs simple tasks until it runs out of data, much as the phonograph plays records until it exhausts its supply. We have not yet told the computer to "think"—that is, to examine the situation and vary its behavior depending on what it finds. Obviously, a phonograph cannot do that.

What we need in our programs is a way to tell the computer to make decisions. We can then place several different sets of instructions (or behavior patterns) in a program and allow the computer to choose among them, depending on the current situation. For example, the computer could terminate a program after a specific number of repetitions, check a student's answer and print one message (CORRECT) or another (WRONG) depending on the answer given, determine whether an inventory level has gone below a reorder point and print a warning if it has, or force a Monopoly player to sell properties when he or she runs out of cash.

The additional BASIC statement that we need is

IF—THEN (execute a specific line number next if a condition is true).

IF—THEN statements let the computer select one of two paths. If the condition following "IF" is true, the computer will branch—that is, execute the line number following "THEN" next. If the condition is false, the computer will plow straight ahead. An IF-THEN statement is like a fork in the road.

The condition following "IF" must be something that is either true or false. You can construct a condition using the relations

= "equals"
> "greater than"
< "less than"
<> "not equal to"

so typical IF-THEN statements are:

```
20 IF I = 1 THEN 60
40 IF X > Y THEN 80
50 IF I < 10 THEN 20
```

Remember, if the condition is false, the computer continues its normal sequence. An IF-THEN statement only directs the computer one way or the other; it does not affect any variables.

Since branches make programs more difficult to follow, we will introduce a BASIC statement that lets us add some explanation to the listings. This statement is

REM (remark)

REM statements do not tell the computer to do anything; the computer merely prints them in the listing. They make programs easier to understand.

Now we can add new features to our programs:

Example 1—arithmetic quiz: Add numbers and print the sums. Limit the number of problems to 4 (see the flowchart in Figure 3-1).

```
 10 DATA 5, 8, 2, 6, 3, 4, 1, 7
 20 LET I = 0
 30 READ A, B
 40 LET C = A + B
 50 LET I = I + 1
 60 PRINT I, "THE SUM OF", A, "AND", B, " = ", C
 70 IF I < 4 THEN 30
999 END
```

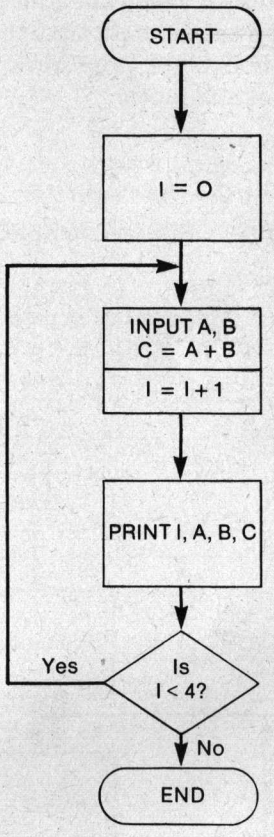


Figure 3-1. Flowchart for addition program with a limited number of iterations

The only change from the previous version is in line 70. If I is less than 4, the computer returns to line 30 and calculates another sum. If I is not less than 4, the computer proceeds sequentially to line 999 and stops. The results are the same as before, except that no error message is printed.

Alternatively we can handle the number of problems (N) as data so that it can have any value.

```
  5 DATA 4
 10 DATA 5, 8, 2, 6, 3, 4, 1, 7
 15 READ N
 20 LET I = 0
 30 READ A, B
 40 LET C = A + B
 50 LET I = I + 1
 60 PRINT I, "THE SUM OF", A, "AND", B, " = ", C
 70 IF I < N THEN 30
999 END
```

You can change the number of problems by changing the value on line 5.

An alternative (see the flowchart in Figure 3-2) is to end the data with a "dummy problem," such as one in which the first value is zero. The following program implements this alternative; note that the dummy problem must include a second value, even though the program never uses it (otherwise, the computer will run out of values and quit).

```
 10 DATA 5, 8, 2, 6, 3, 4, 1, 7, 0, 0
 20 LET I = 0
 30 READ A, B
 32 REM QUIT IF THE FIRST OPERAND IS ZERO
 35 IF A = 0 THEN 999
 40 LET C = A + B
 50 LET I = I + 1
 60 PRINT I, "THE SUM OF", A, "AND", B, " = ", C
 70 GOTO 30
999 END
```

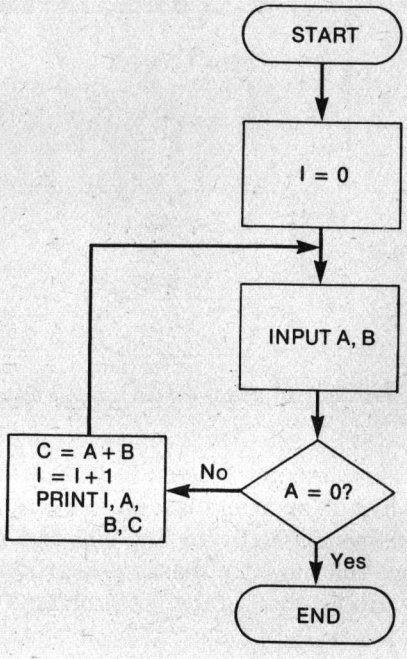


Figure 3-2. Flowchart for addition program with a dummy final problem

Example 2—inventory control: Calculate current inventory levels and mark levels that are at or below a reorder point L.

```
 10 PRINT "ITEM NO   OLD LEVEL   SALES   PURCHASES   NEW LEVEL"
 20 DATA 20, 5, 10, 40, 10, 0, 30, 5, 5
 30 DATA 50, 1, 8, 30, 7, 2, 40, 12, 10
 35 LET L = 30
 40 LET I = 0
 50 READ F, S, P
 60 LET C = F - S + P
 70 LET I = I + 1
 75 REM CHECK IF NEW LEVEL IS ABOVE REORDER POINT
 80 IF C > L THEN 110
 90 PRINT I, F, S, P, C, "REORDER!"
100 GOTO 50
110 PRINT I, F, S, P, C
120 GOTO 50
999 END
```

From the results printed by this program, the user can see at a glance which items are in short supply and must be reordered. Note that we need two "GOTO 50" statements (lines 100 and 120). The first one (line 100) stops the

Figure 3-3. Computer output from an inventory program with reorder warnings.

```
 10 PRINT "ITEM NO    OLD LEVEL    SALES    PURCHASES    NEW LEVEL"
 15 PRINT
 20 DATA 20,5,10,40,10,0,30,5,5
 30 DATA 50,1,8,30,7,2,40,12,10
 35 LET L = 30
 48 LET I = 0
 50 READ F,S,P
 60 LET C = F - S + P
 70 LET I = I + 1
 75 REM CHECK IF NEW LEVEL IS ABOVE REORDER POINT
 80 IF C > L THEN 110
 90 PRINT I,F,S,P,C; "REORDER!"
100 GOTO 50
110 PRINT I,F,S,P,C
120 GOTO 50
999 END

]RUN
ITEM NO          OLD LEVEL        SALES        PURCHASES        NEW LEVEL

1                  20               5              10            25 REORDER!
2                  40              10               0            30 REORDER!
3                  30               5               5            30 REORDER!
4                  50               1               8            57
5                  30               7               2            25 REORDER!
6                  40              12              10            30

?OUT OF DATA ERROR IN 50
```

computer from printing the results twice if a reorder is necessary; remember that the computer works sequentially unless you tell it not to with a GOTO or IF-THEN statement.

Figure 3-3 shows an actual computer listing for this program, followed by the results from running it. We modified line 90 slightly (putting a semicolon after C instead of a comma) to fit the output on one of our computer's lines. We determined the spacing on line 10 by experimentation and added line 15 to leave space vertically between the headings and the data.

An important warning for beginners: Computers do not automatically check to see if the data or the results make sense. In fact the computer has no common sense unless the programmer puts it in the program. The computer does not care if you bought 1,000 hammers at $5 apiece or 5 hammers at $1,000 apiece, nor does it care if your remaining stock of hammers is 125, −350, or 275,000. One famous "computer" error (caused by someone punching an extra column in a card) resulted in a city tax collector assessing a used car at $7 million. In fact, the computer did not make an error; it did precisely what the program said. The computer did not realize that a used car could not be worth $7 million, any more than you would recognize an unreasonable value for some item with which you were totally unfamiliar (such as yak hides or an ounce of gadolinium). The programmer should have had the computer check the data and the results and print warnings if the values were unreasonable.

For example, a current inventory level that is negative clearly makes no sense. The following program prints a warning in that case, alerting the operator to check the data.

```
10 PRINT "ITEM NO   OLD LEVEL   SALES   PURCHASES   NEW LEVEL"
20 DATA 20, 25, 10, 40, 10, 0, 30, 45, 0
30 LET I = 0
40 READ F, S, P
50 LET I = I + 1
60 LET C = F - S + P
65 REM PRINT A WARNING IF STOCK IS NOW NEGATIVE
70 IF C < 0 THEN 100
80 PRINT I, F, S, P, C
90 GOTO 40
100 PRINT I, F, S, P, C, "WARNING—STOCK IS NEGATIVE"
110 GOTO 40
999 END
```

Example 3—Monopoly: Move all players' pieces. Remember that a player who passes Go starts around the board again.

We may represent positions on the Monopoly board by numbers between 1 and 40; Figure 3-4 shows how our numbers correspond to the actual squares. For example, square 1 is Go, square 11 is Jail, and square 21 is Free Parking. A player who crosses square 40 reaches Go and starts around the board again. The movement of pieces around the board is like the movement of the minute hand of a clock that passes 59 and returns to zero. The following program (see Figure 3-5 for a flowchart) updates the positions of the players for one round of moves. P is the previous position of a particular player and M is the move the player rolled on the dice.

1	2	3	4	5	6	7	8	9	10	11
40										12
39										13
38										14
37										15
36										16
35										17
34										18
33										19
32										20
31	30	29	28	27	26	25	24	23	22	21

Figure 3-4. A Monopoly board

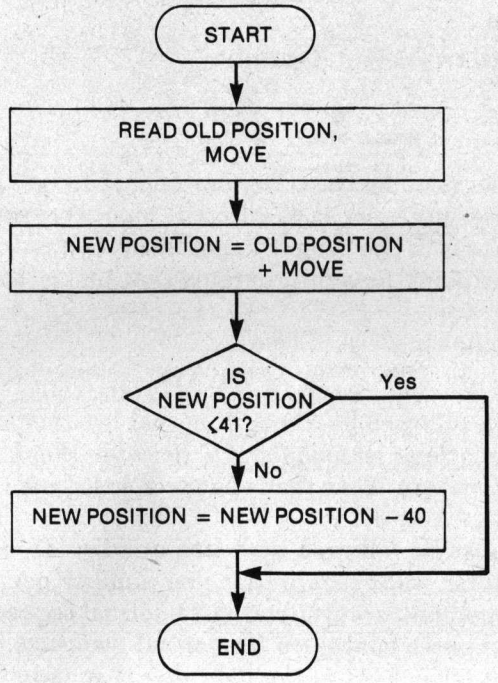


Figure 3-5. Flowchart for a move in Monopoly

The key point here is that the computer must determine if the position P is more than 40. If it is, the computer must subtract 40 from it, since position "41" is Go, the same as position 1. Similarly, minute "61" on a clock is the same as minute 1. After all, going around the board (or the clock) should get you back to the same place (numerically) where you started, just like walking in a circle. Note that we must tell the computer how many squares

there are on the board; it does not know by itself, any more than a baby knows how many minutes there are in an hour.

```
 10 PRINT "PLAYER NO   NEW POSITION"
 20 DATA 37, 10, 21, 9, 35, 6, 12, 3
 30 LET I = 0
 35 REM GET OLD POSITION (P), MOVE (M)
 40 READ P, M
 50 LET I = I + 1
 55 REM FIND NEW POSITION BY ADDING MOVE TO OLD POSITION
 60 LET P = P + M
 65 REM CHECK IF PIECE HAS REACHED OR PASSED GO
 70 IF P < 41 THEN 90
 75 REM IF SO, THEN THE PIECE HAS MOVED AROUND THE BOARD
 77 REM AND WE MUST ADJUST ITS POSITION APPROPRIATELY
 80 LET P = P - 40
 90 PRINT I, P
100 GOTO 40
999 END
```

BRIEF REVIEW

We have introduced two new statements:

IF-THEN	(a very important one)
REM	(a minor one)

IF-THEN statements make the computer appear to be "intelligent," since it can vary its behavior in response to different inputs or results. Note, however, that the intelligence is in the program, not in the computer. REM statements allow us to place explanations in programs that make the programs easier to understand.

Remember the following:

- —Each decision requires an IF-THEN statement.
- —IF must be followed by a condition that is either true or false. You can use the relations = (equals), > (greater than), < (less than), and <> (not equal to). Note that you can't just type "IF X-Y"; you must type "IF X - Y = 0".
- —THEN must be followed by a line number. If the condition is true, the computer will execute that line number next; if the condition is false, the computer will continue its normal sequence of operations.
- —If the computer continues its normal sequence, the condition must have been false. This is the old "dog that didn't bark in the night" logic for Sherlock Holmes fans. The reasoning goes "If an intruder had entered, the dog would have barked. However, the dog did not bark; therefore, no intruder entered."
- —The computer does not use common sense unless the programmer includes it in the program. The computer will not automatically check to see if the data is reasonable or the results are sensible. A computer is like a person without any life experience.
- —The computer can count by simply adding one to a variable at the proper time. But the program must initiate the variable correctly.

INTERACTION

Next we want to let the user interact with the program while it is running. Interaction allows a student to answer questions, an operator to enter data or select options, and a game player to indicate whether he or she wants to purchase a property.

The new BASIC statement that we need is:

INPUT (enter data from the keyboard while the program is running)

An INPUT statement tells the computer to print a question mark and wait for the user to respond. The program can help the user by printing an explanation of the question. The computer will not proceed until the user responds (and presses ENTER or RETURN). Typical sequences involving INPUT statements are:

```
10 PRINT "HOW MANY PLAYERS"
20 INPUT N
```

The PRINT statement explains what information the computer wants. You need not include a question mark in the text since the INPUT statement causes the computer to print one automatically.

```
10 PRINT "WHAT IS THE SUM OF", A, "AND", B
20 INPUT G
```

or alternatively

```
10 PRINT A, " + ", B, " = "
20 INPUT G
```

```
10 PRINT "DO YOU WANT TO CONTINUE (ENTER 0 FOR NO, 1 FOR YES)"
20 INPUT D
30 IF D = 0 THEN 999
40 IF D <> 1 THEN 10
```

The last series of statements allows the user to stop the program. We have coded the valid answers in a simple way (0 for NO, 1 for YES), included a PRINT statement that explains what information the computer wants, and (in line 40) forced the computer to repeat the question if the user accidentally enters an invalid answer (something other than 0 or 1). Remember that "<>" means "not equal to."

Example 1—arithmetic quiz: Now at last we can make the addition program into a real quiz.

```
10 DATA 5, 8, 2, 6, 3, 4, 1, 7
20 LET I = 0
25 REM READ TWO NUMBERS AND DETERMINE THE CORRECT SUM
30 READ A, B
40 LET C = A + B
50 LET I = I + 1
55 REM POSE THE PROBLEM TO THE STUDENT
60 PRINT A, " + ", B, " = "
65 REM LET THE STUDENT ENTER AN ANSWER
70 INPUT G
75 REM CHECK IF THE ANSWER IS CORRECT AND TELL THE STUDENT
```

```
 80 IF C = G THEN 110
 90 PRINT "THAT'S WRONG"
100 GOTO 30
110 PRINT "THAT'S CORRECT"
120 GOTO 30
999 END
```

We can add grading by simply counting the number of correct answers (we call it R). Figure 3-6 contains a flowchart of this program and Figure 3-7 contains an actual computer listing, combined with an interactive session. The student's answers follow the question marks.

```
 10 DATA 5, 8, 2, 6, 3, 4, 1, 7
 20 LET I = 0
 22 LET R = 0
 25 REM READ TWO NUMBERS AND DETERMINE THE CORRECT SUM
 30 READ A, B
 40 LET C = A + B
 50 LET I = I + 1
 55 REM POSE THE PROBLEM TO THE STUDENT
 60 PRINT A, " + ", B, " = "
 65 REM LET THE STUDENT ENTER AN ANSWER
 70 INPUT G
 75 REM CHECK IF THE ANSWER IS CORRECT AND TELL THE STUDENT
 80 IF C = G THEN 110
 90 PRINT "THAT'S WRONG"
100 GOTO 130
110 PRINT "THAT'S CORRECT"
115 REM ADD 1 TO THE NUMBER OF CORRECT ANSWERS (R)
120 LET R = R + 1
125 REM TELL THE STUDENT HOW HE OR SHE IS DOING
130 PRINT "YOU HAVE", R, "RIGHT ANSWERS OUT OF", I
140 GOTO 30
999 END
```

Example 2—inventory control: Make the inventory program interactive, allowing the operator to enter the data from the keyboard. Here the operator must enter several values separated by commas.

```
 10 LET I = 0
 20 LET I = I + 1
 25 REM ASK THE OPERATOR TO ENTER THE DATA FOR THIS ITEM
 30 PRINT "ITEM NUMBER", I
 40 PRINT "ENTER OLD STOCK, SALES, AND PURCHASES IN THAT
      ORDER"
 50 PRINT "PLACE COMMAS BETWEEN VALUES"
 60 INPUT F, S, P
 65 REM CALCULATE THE NEW LEVEL AND PRINT IT
 70 LET C = F - S + P
 80 PRINT "NEW LEVEL IS", C
 90 GOTO 20
999 END
```

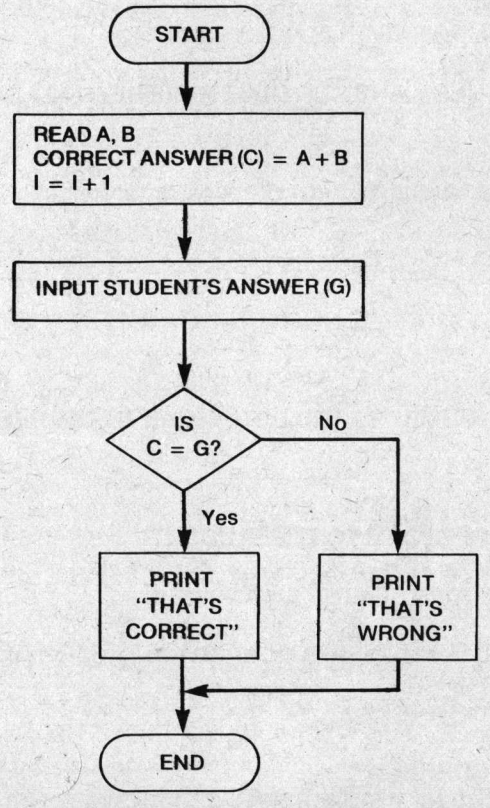


Figure 3-6. Flowchart for arithmetic quiz with grading

Note the difference between a READ statement that obtains values from DATA statements and an INPUT statement that asks the operator to enter values while the program is running.

This program will run forever, but we can let the operator escape from it by using a sequence similar to the one we mentioned.

```
10 LET I = 0
20 LET I = I + 1
25 REM ASK THE OPERATOR TO ENTER THE DATA FOR THIS ITEM
30 PRINT "ITEM NUMBER", I
40 PRINT "ENTER OLD STOCK, SALES, AND PURCHASES IN THAT
     ORDER"
50 PRINT "PLACE COMMAS BETWEEN VALUES"
60 INPUT F, S, P
65 REM CALCULATE THE NEW LEVEL AND PRINT IT
70 LET C = F - S + P
80 PRINT "NEW LEVEL IS", C
85 REM ASK OPERATOR IF HE OR SHE WISHES TO CONTINUE
90 PRINT "DO YOU WANT TO CONTINUE (ENTER 0 FOR NO, 1 FOR YES)"
100 INPUT D
105 REM PROCEED TO NEXT ITEM IF ANSWER IS YES
```

```
110 IF D = 1 THEN 20
115 REM ASK AGAIN IF ANSWER IS NOT YES OR NO
120 IF D <> 0 THEN 90
125 REM IF WE GOT HERE, ANSWER MUST HAVE BEEN NO, SO STOP
999 END
```

Figure 3-7. Computer output from an arithmetic quiz program with grading

```
]LIST

 10 DATA 5,8,2,6,3,4,1,7
 20 LET I = 0
 22 LET R = 0
 25 REM READ TWO NUMBERS AND DETERMINE THE CORRECT SUM
 30 READ A,B
 40 LET C = A + B
 50 LET I = I + 1
 55 REM POSE THE PROBLEM TO THE STUDENT
 60 PRINT A;" + ";B;" = "
 65 REM LET THE STUDENT ENTER AN ANSWER
 70 INPUT G
 75 REM CHECK IF THE ANSWER IS CORRECT AND TELL THE STUDENT
 80 IF C = G THEN 110
 90 PRINT "THAT'S WRONG"
100 GOTO 130
110 PRINT "THAT'S CORRECT"
115 REM ADD 1 TO THE NUMBER OF CORRECT ANSWERS (R)
120 LET R = R + 1
125 REM TELL THE STUDENT HOW HE OR SHE IS DOING
130 PRINT "YOU HAVE ";R;" RIGHT ANSWERS OUT OF ";I
140 GOTO 30
999 END

]RUN
5 + 8 =
?7
THAT'S WRONG
YOU HAVE 0 RIGHT ANSWERS OUT OF 1
2 + 6 =
?8
THAT'S CORRECT
YOU HAVE 1 RIGHT ANSWERS OUT OF 2
3 + 4 =
?7
THAT'S CORRECT
YOU HAVE 2 RIGHT ANSWERS OUT OF 3
1 + 7 =
?8
THAT'S CORRECT
YOU HAVE 3 RIGHT ANSWERS OUT OF 4

?OUT OF DATA ERROR IN 30
```

Figure 3-8 shows an actual computer listing and an interactive session for this program. The operator's entries, as in Figure 3-7, follow the question marks. At the bottom of Figure 3-8, the operator has erroneously responded to the question of whether the program should continue with an answer of 5. The computer then repeats the question and stops the program when the answer is 0 (NO). A programmer should never assume that the user will provide a reasonable answer. People often become confused and make typing errors; the program should allow for this.

ASSIGNING NUMERICAL VALUES

To allow the operator to stop the program, we arbitrarily assigned numerical values to the answers; we decided that

0 = response of "NO"
1 = response of "YES"
Other responses result in a repetition of the question

We could just as easily have said that 6500 meant NO and 8237 meant YES, but 0 and 1 are shorter and simpler. If there were three possible answers, we could assign them the values 0, 1, 2 or 1, 2, 3. Most personal computers have versions of BASIC that handle data consisting of typed characters (using so-called *string variables*); a program for such a computer can read a typed answer and determine if it is YES or NO. Other kinds of data, however, still require the arbitrary assignment of values.

Example 3—Monopoly: The game involves a large amount of non-numerical data. Of course, we have already assigned numbers (1 to 40) to the squares on the board. Other ideas that we must handle include

—Whether a square can be purchased: for example, players cannot buy Go, Jail, Chance, Community Chest, or Income Tax.
—The various categories of properties: some squares are railroads, others are utilities, while still others belong to groups of two or three. Remember that the rent increases if a player owns an entire group of properties.
—Whether anyone owns a property. If no one owns it, the player who lands on it can buy it. If someone owns it, the player who lands on it must pay rent (unless he or she owns it).
—Whether a player is eligible to move. A player may lose a turn or spend time in Jail.
—The condition of a property. Properties may be mortgaged and some can have houses or a hotel built on them.

We can handle the ownership of a property by assigning the following values to a variable W:

If no one owns the property, W = 0.

If player J owns the property, W = J; for example, if player 2 owns the property, W = 2.

Figure 3-8. Computer output from an interactive inventory program that the operator can stop

```
]LIST

 10 LET I = 0
 20 LET I = I + 1
 25 REM ASK THE OPERATOR TO ENTER THE DATA FOR THIS ITEM
 30 PRINT "ITEM NUMBER ";I
 40 PRINT "ENTER OLD STOCK, SALES, AND PURCHASES IN THAT ORDER"
 50 PRINT "PLACE COMMAS BETWEEN VALUES"
 60 INPUT F,S,P
 65 REM CALCULATE THE NEW LEVEL AND PRINT IT
 70 LET C = F - S + P
 80 PRINT "NEW LEVEL IS ";C
 85 REM ASK OPERATOR IF HE OR SHE WISHES TO CONTINUE
 90 PRINT "DO YOU WANT TO CONTINUE (ENTER 0 FOR NO, 1 FOR YES)"
100 INPUT D
105 REM PROCEED TO NEXT ITEM IF ANSWER IS YES
110 IF D = 1 THEN 20
115 REM ASK AGAIN IF ANSWER IS NOT YES OR NO
120 IF D <> 0 THEN 90
125 REM IF WE GOT HERE, ANSWER MUST HAVE BEEN NO, SO STOP
999 END


]RUN
ITEM NUMBER 1
ENTER OLD STOCK, SALES, AND PURCHASES IN THAT ORDER
PLACE COMMAS BETWEEN VALUES
?123,45,67
NEW LEVEL IS 145
DO YOU WANT TO CONTINUE (ENTER 0 FOR NO, 1 FOR YES)
?1
ITEM NUMBER 2
ENTER OLD STOCK, SALES, AND PURCHASES IN THAT ORDER
PLACE COMMAS BETWEEN VALUES
?145,67,8
NEW LEVEL IS 86
DO YOU WANT TO CONTINUE (ENTER 0 FOR NO, 1 FOR YES)
?1
ITEM NUMBER 3
ENTER OLD STOCK, SALES, AND PURCHASES IN THAT ORDER
PLACE COMMAS BETWEEN VALUES
?21,34,10
NEW LEVEL IS -3
DO YOU WANT TO CONTINUE (ENTER 0 FOR NO, 1 FOR YES)
?5
DO YOU WANT TO CONTINUE (ENTER 0 FOR NO, 1 FOR YES)
?0
```

The following program determines if a player must pay rent when he or she lands on a square. J is the player's number, P is the new position, and W is the ownership of square P. The player does not pay rent if no one owns the square or if he or she owns the square. Figure 3-9 contains a flowchart of this procedure.

```
 10 DATA 4, 25, 3
 15 REM READ PLAYER NUMBER, POSITION, OWNERSHIP
 20 READ J, P, W
 25 REM PAY NO RENT IF NO ONE OWNS THIS SQUARE
 30 IF W = 0 THEN 70
 35 REM PAY NO RENT IF THE PLAYER OWNS THE SQUARE
 40 IF W = J THEN 70
 45 REM SOMEONE ELSE OWNS THE SQUARE — PAY UP
 50 PRINT "PLAYER", J, "MUST PAY RENT TO PLAYER", W
 60 GOTO 999
 65 REM NO RENT TO PAY
 70 PRINT "PLAYER", J, "DOES NOT HAVE TO PAY RENT"
999 END
```

The IF-THEN statement on line 30 causes a branch if no one owns the property (W = 0). The IF-THEN statement on line 40 causes a branch if the player has landed on a square that he or she already owns (W = J, the

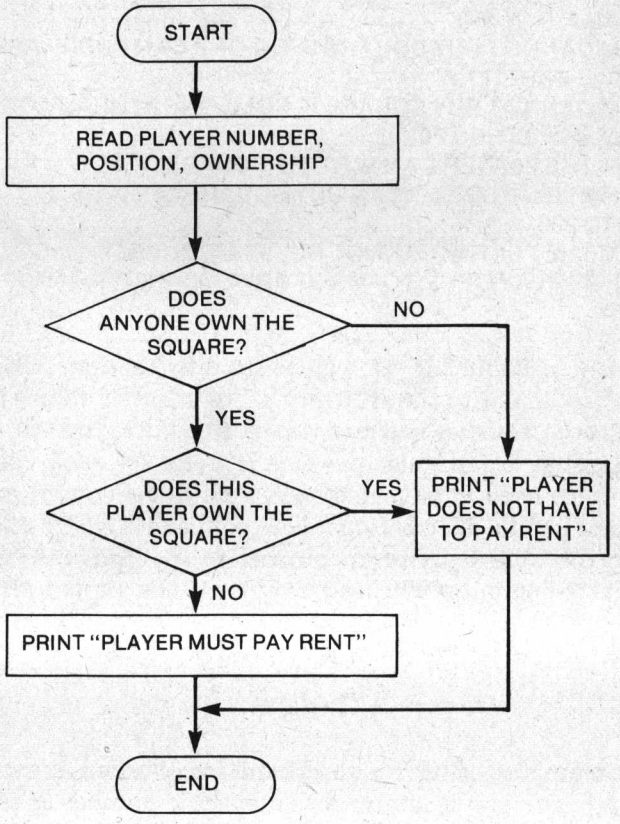


Figure 3-9. Flowchart for determining if a Monopoly player must pay rent

number of the player who has just moved). The GOTO on line 60 concludes the procedure if the player must pay rent; we do not want the computer to execute line 70 and confuse everybody by printing two contradictory messages (PLAYER 4 MUST PAY RENT TO PLAYER 3 and PLAYER 4 DOES NOT HAVE TO PAY RENT).

Similarly we can write a program that tells a player if he or she can buy a square. A player can buy the square he or she lands on if it can be purchased and if no one already owns it. We will handle whether the square can be purchased by assigning the following values to a variable B:

If the square cannot be purchased, B = 0

If the square can be purchased, B = 1

A value of B is permanently attached to each square, whereas the value of W, for a square that can be purchased, must be changed when the square is sold. The following program tells a player whether he or she can buy a square. J is the ownership of square P. The player does not pay rent if no one owns the square or if he or she owns the square. Figure 3-9 contains a flowchart of this procedure.

```
 10 DATA 4, 25, 3, 0
 15 REM READ PLAYER NUMBER, POSITION, OWNERSHIP,
 17 REM  PURCHASABILITY
 20 READ J, P, W, B
 25 REM CANNOT BUY SQUARE IF NO ONE CAN PURCHASE IT
 30 IF B = 0 THEN 70
 35 REM CANNOT BUY SQUARE IF SOMEONE ALREADY OWNS IT
 40 IF W <> 0 THEN 70
 45 REM THE PLAYER CAN BUY THE SQUARE
 50 PRINT "PLAYER", J, "CAN BUY SQUARE", P
 60 GOTO 999
 65 REM THE PLAYER CANNOT BUY THE SQUARE
 70 PRINT "PLAYER", J, "CANNOT BUY SQUARE", P
999 END
```

The arbitrary assignment of values to abstractions like "owned" or "purchasable" is similar to the arbitrary assignment of values to letter grades (A = 3, B = 2, etc.), finishes in a race (1st = 10 points, 2nd = 8 points, etc.), or votes for or against a particular position (1 point for each vote that favored our position, 0 for each vote that opposed it). Some computer languages let the programmer handle abstractions more conveniently; for example, the programmer may be able to refer to properties as "purchasable" or "unpurchasable" and the computer will assign values to the various alternatives.

LOOPS

Most computer languages have a shorthand form (called a *loop*) that tells the computer to repeat a set of statements a specific number of times. We have, of course, managed to implement loops using counts and IF-THEN statements. The shorthand form, however, is simpler and more general. The new BASIC statement that we need is

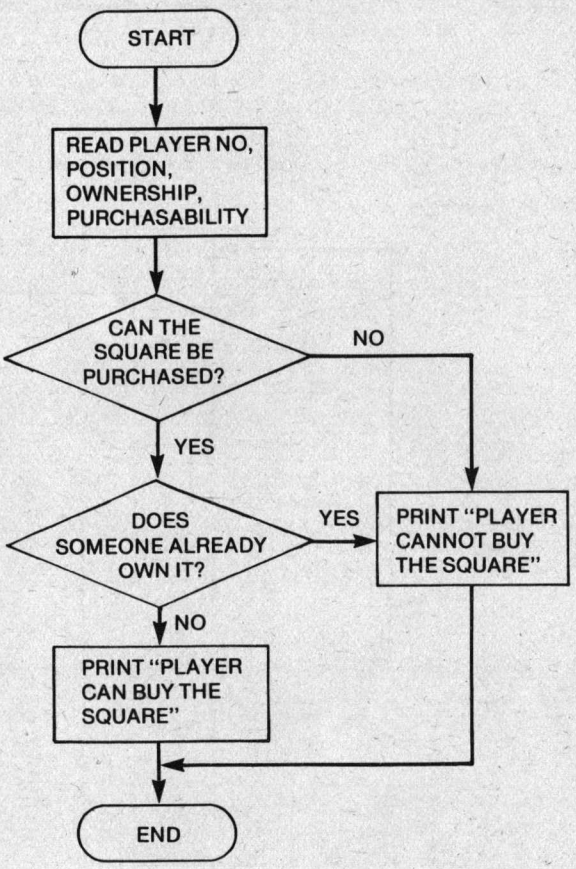


Figure 3-10. Flowchart for determining if a Monopoly player can buy a property

FOR-NEXT (repeat the statements between FOR and NEXT for values of the specified variable starting at a specified initial value and ending at a specified final value)

A typical program using a **FOR-NEXT statement** is the following summation program. It adds together a series of numbers and prints the sum. Figure 3-11 contains a flowchart of the program.

```
 10 DATA 5, 7, 3, 8, 2, 1, 5, 4, 3, 6
 15 REM INITIATE THE SUM TO ZERO
 20 LET S = 0
 25 REM FORM THE SUMMATION, ADDING ONE NUMBER AT A TIME
 30 FOR I = 1 TO 10
 35 REM GET THE NEXT NUMBER AND ADD IT TO THE SUM
 40 READ T
 50 LET S = S + T
 60 NEXT I
 70 PRINT "THE SUM", S
999 END
```

The FOR (on line 30) and the NEXT (on line 60) form a unit, since the same variable name (I) appears in both. The computer repeats all the statements between FOR and NEXT for values of I (the loop variable) as specified

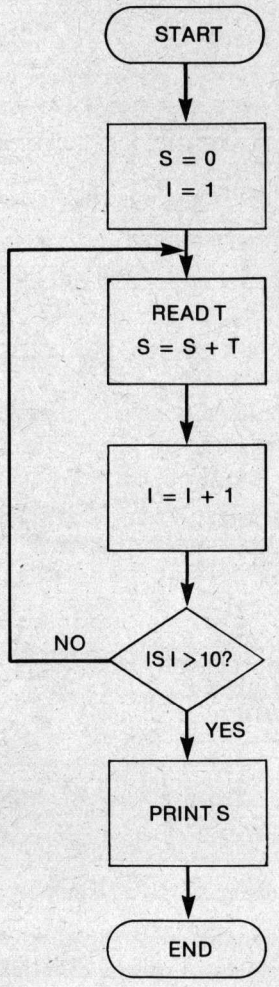


Figure 3-11. Flowchart for summation program

after "FOR". The form in which the values are specified is = L TO U, where L is the lower limit or initial value and U is the upper limit or final value. When the computer reaches NEXT, it automatically adds 1 to I and checks to see if I now exceeds the final value. If not, the computer returns to the start of the loop. Figure 3-12 is a flowchart of the execution for a FOR–NEXT statement. The loop variable can have any name, but you should be careful not to confuse loop variables with ordinary program variables.

Let us now examine in detail how the computer executes our example. When it reaches the FOR statement, the computer gives I the specified initial value (1). So the values at various points in the program are as follows:

BEFORE ENTERING THE LOOP

S = 0

1ST ITERATION (after line number 50)

I = 1
T = 5
S = S + 5 = 5

2ND ITERATION (after line number 50)

I = 2
T = 7
S = S + 7 = 12

3RD ITERATION (after line number 50)

I = 3
T = 3
S = S + 3 = 15

•
•
•

10TH ITERATION (after line number 50)

I = 10
T = 6
S = S + 6 = 44

Note that the computer executes the loop for the final value of I.

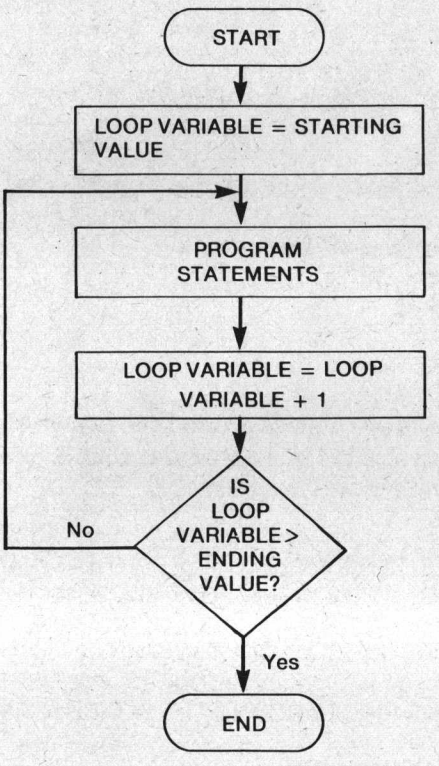


Figure 3-12. Flowchart for a BASIC FOR-NEXT statement

We can use FOR-NEXT statements to simplify our earlier programs.

Example 1—arithmetic quiz: Add numbers and print the sums. Limit the number of problems to 4 (see the flowchart in Figure 3-1).

```
  5 DATA 4
 10 DATA 5, 8, 2, 6, 3, 4, 1, 7
 12 REM READ THE NUMBER OF PROBLEMS (N)
 15 READ N
 20 FOR I = 1 TO N
 25 REM READ TWO NUMBERS AND DETERMINE THEIR SUM
 30 READ A, B
 40 LET C = A + B
 50 PRINT I, "THE SUM OF", A, "AND", B, " = ", C
 60 NEXT I
999 END
```

This program does not need the statements that set I to 0, add 1 to it, and compare its value with N since the FOR-NEXT statement handles those details automatically.

Example 2—inventory control: Calculate current inventory levels (C) from former levels (F), sales (S), and purchases (P) for 6 items.

```
 10 PRINT "ITEM NO   OLD LEVEL   SALES   PURCHASES   NEW LEVEL"
 15 DATA 6
 20 DATA 20, 5, 10, 40, 10, 0, 30, 5, 5
 30 DATA 50, 1, 8, 30, 7, 2, 40, 12, 10
 32 REM READ THE NUMBER OF ITEMS (N)
 35 READ N
 40 FOR I = 1 TO N
 45 REM READ FORMER STOCK, SALES, PURCHASES
 50 READ F, S, P
 55 REM DETERMINE CURRENT STOCK
 60 LET C = F - S + P
 70 PRINT I, F, S, P, C
 80 NEXT I
999 END
```

Example 3—Monopoly: Move a piece around the board. Here P is the position of the piece, M is the move rolled in a particular round, and N is the total number of moves. The piece starts at Go (square 1).

```
 10 DATA 10
 20 DATA 5, 9, 11, 6, 12, 10, 4, 7, 9, 6
 25 REM READ THE TOTAL NUMBER OF MOVES N
 30 READ N
 35 REM START THE PIECE AT GO (SQUARE 1)
 40 LET P = 1
 45 REM MOVE THE PIECE AROUND THE BOARD N TIMES
 50 FOR I = 1 TO N
 55 REM READ THE NEXT MOVE
 60 READ M
 65 REM UPDATE THE PIECE'S POSITION
```

```
 70 LET P = P + M
 75 REM CHECK IF PIECE HAS REACHED OR PASSED GO
 80 IF P < 41 THEN 100
 85 REM IF SO, THEN THE PIECE HAS MOVED AROUND THE BOARD
 87 REM AND WE MUST ADJUST ITS POSITION APPROPRIATELY
 90 LET P = P - 40
100 PRINT "MOVE NUMBER", I, "POSITION", P
110 NEXT I
999 END
```

The FOR-NEXT statement does not actually give us any new capabilities, but it makes programs easier to understand since it indicates repetitions.

BRIEF REVIEW

We have introduced these new statements:

FOR-NEXT
INPUT

Remember the following:

- —INPUT allows the user to enter data and control the program while it is running. The program must include the PRINT statements that explain the questions.
- —The computer will wait for the user's response. The program must, however, check to see if the response is sensible.
- —One way to allow the computer to handle non-numerical data is to assign arbitrary numerical values to the alternatives. The program can then deal with the numerical values.
- —The FOR-NEXT statement simplifies the repetition of part of a program. It automatically initiates the counter, increases its value after each iteration, and compares it to the maximum value.

PROCESSING SETS OF DATA

We have so far ignored the problem of handling many elements of the same type. For example:

- —*In the arithmetic quiz program:* We might want to assign grades to each student, chart each student's progress, determine the average grade and average progress, and perhaps see how a particular student or the entire class did on a particular problem or group of problems. We need to handle groups of problems, groups of grades, and groups of students.
- —*In the inventory program:* We might want to determine the overall value of the inventory; total sales by item, department, or store; the fastest and slowest moving items; and items that must be reordered or are overstocked.

—*In the Monopoly program:* .We must keep track of all the players' positions, cash, and properties. We must also credit rents to the appropriate players and handle Chance and Community Chest with their fines, penalties, and rewards.

We need to be able to refer to one element of a set, e.g., the score of the 8th student, the sales of the 27th item, or the position of the 3rd player. The way in which we make these references in BASIC is by placing the element number in parentheses after the name of the set. That is, we refer to the Ith element of the set A as A(I). A is often referred to as an *array* and I as an *index.* Thus

—If S is the set of weekly sales of various items, S(5) is the sales of item number 5, S(12) of item number 12, and S(L) of item number L.
—If G is the set of student grades, G(1) is the grade for student number 1, G(8) for student number 8, and G(K) for student number K.
—If P is the set of the positions of players on the Monopoly board, P(2) is the position of player number 2, P(4) of player number 4, and P(J) of player number J.

We must arbitrarily assign a number to each item, student, or player.

We need an additional BASIC statement that tells the computer how much memory to assign to the set. This statement is

DIM (dimension: specify the maximum number of elements in a set)

The form is

```
10 DIM A(25), B(10), C(30)
```

DIM statements only tell the computer how large the sets are; they do not affect the execution of the program unless you omit a set or do not specify enough elements. You must specify the maximum number of elements in each set in a DIM statement. You can avoid accidental omissions by adding the DIM statements after you finish the program; list the sets you have used and place them all in DIM statements. You can avoid exceeding the maximum number of elements by always allowing more than you actually need. DIM statements can appear anywhere in a program, but programmers usually place them at the beginning to simplify changes. DIM statements need line numbers even though they are not executed.

Let us look at some simple examples.

Example 1—arithmetic quiz: Enter the scores and calculate the average score. Figure 3-13 contains the flowchart. The procedure, of course, is to sum the scores and divide by the number of students. Here S(I) is the score for the Ith student, T is the sum, N is the number of students, and A is the average.

```
10 DIM S(25)
20 DATA 5
25 REM READ NUMBER OF STUDENTS
30 READ N
35 REM INITIATE SUM TO ZERO
40 LET T = 0
50 FOR I = 1 TO N
55 REM ASK FOR NEXT SCORE
```

```
 60 PRINT "ENTER SCORE FOR STUDENT", I
 70 INPUT S(I)
 75 REM ADD SCORE INTO TOTAL
 80 LET T = T + S(I)
 90 NEXT I
 95 REM CALCULATE AVERAGE FROM TOTAL, NUMBER OF STUDENTS
100 LET A = T/N
110 PRINT "AVERAGE GRADE IS", A
999 END
```

At the end of the program, the computer still remembers all the scores, since it saved them in separate places in its memory, rather than writing one on top of another. So we can easily extend the program to let the teacher enter the passing score and have the computer print a list of who passed and who failed. Figure 3-14 is a flowchart of the additional procedure (lines 120 through 200) and Figure 3-15 contains an actual computer listing and an interactive session. We modified lines 170 and 190 somewhat to make the results clearer.

```
120 PRINT "ENTER PASSING SCORE"
130 INPUT P
```

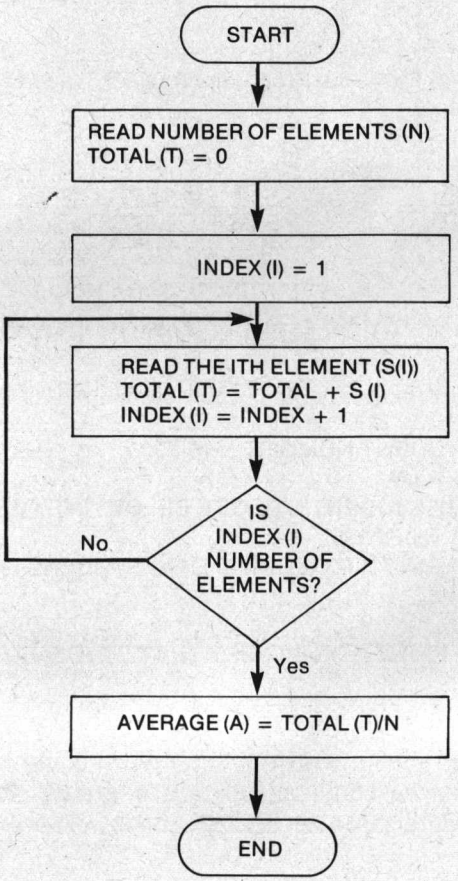


Figure 3-13. Flowchart for finding an average

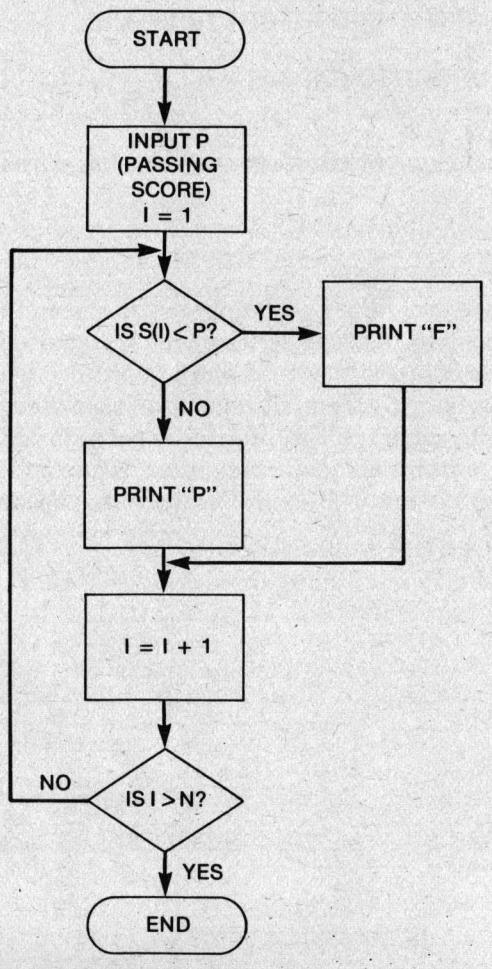


Figure 3-14. Flowchart for determining which students passed and which failed

```
140 PRINT "STUDENT NUMBER   P/F"
150 FOR I = 1 TO N
155 REM IS THE STUDENT'S SCORE BELOW THE PASSING MARK?
160 IF S(I) < P THEN 190
165 REM NO, PRINT P TO SIGNIFY THAT STUDENT PASSED
170 PRINT I, "P"
180 GOTO 200
185 REM YES, PRINT F TO SIGNIFY THAT STUDENT FAILED
190 PRINT I, "F"
200 NEXT I
```

As long as all the grades are still available, we could analyze them in other ways. For example, we could assign letter grades, arrange the scores in numerical order, or determine the highest and lowest scores.

Example 2—inventory control: Let the operator enter purchase orders with item number followed by quantity bought. A dummy item number (−1) signifies that the operator has entered all the purchase orders. Figure 3-16 is

```
]LIST

 10 DIM S(25)
 20 DATA 5
 25 REM READ NUMBER OF STUDENTS
 30 READ N
 35 REM INITIATE SUM TO ZERO
 40 LET T = 0
 50 FOR I = 1 TO N
 55 REM ASK FOR NEXT SCORE
 60 PRINT "ENTER SCORE FOR STUDENT";I
 70 INPUT S(I)
 75 REM ADD SCORE INTO TOTAL
 80 LET T = T + S(I)
 90 NEXT I
 95 REM CALCULATE AVERAGE FROM TOTAL, NUMBER OF STUDENTS
100 LET A = T / N
110 PRINT "AVERAGE GRADE IS ";A
120 PRINT "ENTER PASSING SCORE"
130 INPUT P
140 PRINT "STUDENT NUMBER P/F"
150 FOR I = 1 TO N
160 IF S(I) < P THEN 190
165 REM NO, PRINT P TO SIGNIFY THAT STUDENT PASSED
170 PRINT I, "P"
180 GOTO 200
185 REM YES, PRINT F TO SIGNIFY THAT STUDENT FAILED
190 PRINT I," F"
200 NEXT I
999 END

]RUN
ENTER SCORE FOR STUDENT 1
?56
ENTER SCORE FOR STUDENT 2
?89
ENTER SCORE FOR STUDENT 3
?67
ENTER SCORE FOR STUDENT 4
?79
ENTER SCORE FOR STUDENT 5
?65
AVERAGE GRADE IS 71.2
ENTER PASSING SCORE
?76
STUDENT NUMBER P/F
1                F
2            P
3                F
4            P
5                F
```

Figure 3-15. Computer output from an interactive grading program that provides pass/fail information

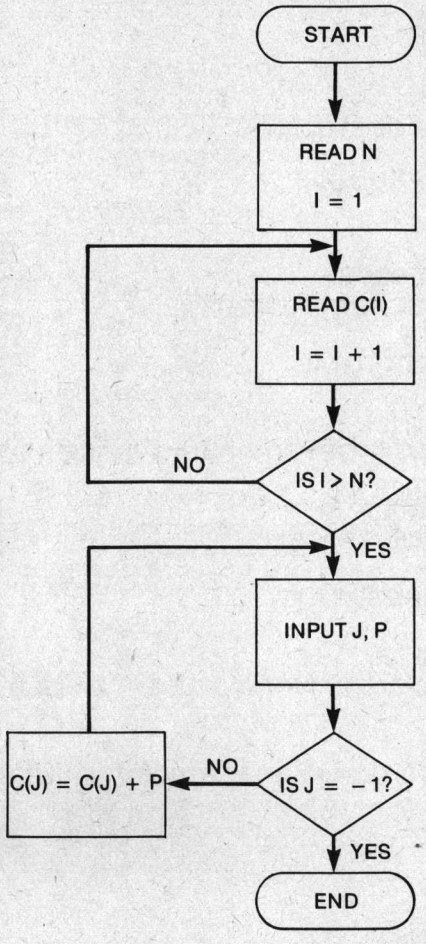


Figure 3-16. Flowchart for letting operator enter purchase orders with item numbers

a flowchart of the procedure and Figure 3-17 contains a computer listing and the record of an interactive session.

```
 10 DIM C(25)
 20 DATA 6
 30 DATA 20, 40, 30, 50, 30, 40
 35 REM READ NUMBER OF ITEMS
 40 READ N
 45 REM READ PREVIOUS STOCKS FOR ALL ITEMS
 50 FOR I = 1 TO N
 60 READ C(I)
 70 NEXT I
 75 REM ASK OPERATOR TO ENTER A PURCHASE ORDER
 80 PRINT "ENTER NEXT PURCHASE ORDER"
 90 PRINT "ITEM NUMBER, QUANTITY — ENTER -1, 0 IF DONE"
100 INPUT J, P
105 REM CHECK IF OPERATOR IS FINISHED (ITEM NUMBER = -1)
110 IF J = -1 THEN 140
115 REM OTHERWISE ADD QUANTITY PURCHASED TO CORRECT STOCK
```

```
120 LET C(J) = C(J) + P
130 GOTO 80
135 REM PRINT FINAL STOCKS FOR ALL ITEMS
140 PRINT "ITEM NO   FINAL STOCK"
150 FOR I = 1 TO N
160 PRINT I, C(I)
170 NEXT I
999 END
```

The operator can enter many different purchase orders for a single item. The orders need not be arranged by item number. Instead, the operator tells the computer (by entering J) which item has been bought. The computer adds the quantity purchased to the previous stock of the Jth item, i.e., to C(J). The operator must enter a dummy item number (−1), along with a quantity, to tell the computer that all purchase orders have been entered. Note that the operator must type both an item number and a quantity, even though the computer never uses the quantity for the dummy item number.

Figure 3-17. Computer output from an interactive inventory program that handles unsorted purchase orders

```
]LIST

 10 DIM C(25)
 20 DATA 6
 30 DATA 20,40,30,50,30,40
 35 REM READ NUMBER OF ITEMS
 40 READ N
 45 REM READ PREVIOUS STOCKS FOR ALL ITEMS
 50 FOR I = 1 TO N
 60 READ C(I)
 70 NEXT I
 75 REM ASK OPERATOR TO ENTER PURCHASE ORDER
 80 PRINT "ENTER NEXT PURCHASE ORDER"
 90 PRINT "ITEM NUMBER, QUANTITY — ENTER -1,0 IF DONE"
100 INPUT J,P
105 REM CHECK IF OPERATOR IS FINISHED (ITEM NUMBER = -1)
110 IF J = -1 THEN 140
115 REM OTHERWISE ADD QUANTITY PURCHASED TO CORRECT STOCK
120 LET C(J) = C(J) + P
130 GOTO 80
135 REM PRINT FINAL STOCKS FOR ALL ITEMS
140 PRINT "ITEM NO    FINAL STOCK"
150 FOR I = 1 TO N
160 PRINT I,C(I)
170 NEXT I
999 END


]RUN
ENTER NEXT PURCHASE ORDER
ITEM NUMBER, QUANTITY — ENTER -1,0 IF DONE
?1,45
```

```
ENTER NEXT PURCHASE ORDER
ITEM NUMBER, QUANTITY — ENTER – 1,0 IF DONE
?2,32
ENTER NEXT PURCHASE ORDER
ITEM NUMBER, QUANTITY — ENTER – 1,0 IF DONE
?3,5
ENTER NEXT PURCHASE ORDER
ITEM NUMBER, QUANTITY — ENTER – 1,0 IF DONE
?4,63
ENTER NEXT PURCHASE ORDER
ITEM NUMBER, QUANTITY — ENTER – 1,0 IF DONE
?5,21
ENTER NEXT PURCHASE ORDER
ITEM NUMBER, QUANTITY — ENTER – 1,0 IF DONE
?6,10
ENTER NEXT PURCHASE ORDER
ITEM NUMBER, QUANTITY — ENTER – 1,0 IF DONE
? – 1,0
ITEM NO          FINAL STOCK
1                 65
2                 72
3                 35
4                 113
5                 51
6                 50
```

Example 3—Monopoly: Start the game. Ask the operator to enter the number of players. Place all the pieces at Go (square 1) and give each player his or her initial $1,500. Assume that P(I) is the position of the Ith player and C(I) is the Ith player's cash holding.

```
 10 DIM P(20), C(20)
 15 REM ASK OPERATOR TO ENTER NUMBER OF PLAYERS
 20 PRINT "HOW MANY PLAYERS"
 30 INPUT N
 35 REM START ALL PLAYERS AT GO (SQUARE 1) WITH $1,500
 40 FOR I = 1 TO N
 50 LET P(I) = 1
 60 LET C(I) = 1500
 70 PRINT "PLAYER NO", I, "POSITION", P(I), "CASH", C(I)
 80 NEXT I
999 END
```

At the end of this startup routine, the computer remembers all the players' cash holdings and positions. We could now start the game, generating moves for all the players and allowing them to buy and sell properties, pay rent, draw cards from Chance or Community Chest, and perform other transactions. Keeping all the positions in a set would allow us to draw a board at the end of each round and show where all the pieces were. Keeping all the cash holdings in a set would allow us to put them in numerical order and show who was ahead. We could also perform transactions involving more than one player. For example, if a player lands on a square belonging to someone else, he

or she must pay rent to the property owner. The renter's cash holding decreases and the landlord's increases by the amount of the rent.

Even more complicated transactions result from the Chance and Community Chest cards that order one player to pay all the others or allow one player to collect from all the others.

RANDOM NUMBERS

Most versions of BASIC can produce random numbers—that is, numbers that have an equal chance to take on any one of their possible values. Typical real-life random numbers are the outcome of rolling an honest die (equally likely to be 1, 2, 3, 4, 5, or 6) or the outcome of flipping an honest coin (equally likely to be heads or tails, which we can refer to as 0 or 1, for example).

The way in which one produces random numbers in BASIC varies from computer to computer, but we will assume that the statement

```
Y = RND(X)
```

makes the value of Y a random whole number (integer) between 1 and X. Thus

```
Y = RND(6)
```

simulates the rolling of an honest die by giving Y a value that is equally likely to be 1, 2, 3, 4, 5, or 6. We will use RND to generate random problems for our arithmetic quiz and to simulate the rolling of dice in the Monopoly game.

Example 1—arithmetic quiz: Generate addition problems randomly, allowing each number to take on any value between 1 and 9.

```
 10 DATA 5
 15 REM READ NUMBER OF PROBLEMS
 20 READ N
 30 FOR I = 1 TO N
 35 REM GENERATE A PROBLEM RANDOMLY
 40 LET A = RND(9)
 50 LET B = RND(9)
 60 LET C = A + B
 65 REM POSE THE PROBLEM TO THE STUDENT
 70 PRINT A, "+", B, "="
 75 REM LET THE STUDENT ENTER AN ANSWER
 80 INPUT G
 85 REM CHECK IF THE ANSWER IS CORRECT AND TELL THE STUDENT
 90 IF C = G THEN 120
100 PRINT "THAT'S WRONG"
110 GOTO 130
120 PRINT "THAT'S CORRECT"
130 NEXT I
999 END
```

RND produces a different random number each time you use it; A and B will not necessarily have the same value even though we have set both of them to RND(9).

Example 2—Monopoly: Move a player around the board, starting at Go (square 1) and generating the moves randomly as if the player had rolled two dice.

```
 10 DATA 20
 15 READ NUMBER OF MOVES
 20 READ N
 25 START PLAYER AT GO (SQUARE 1)
 30 LET P = 1
 35 REM MOVE THE PLAYER AROUND THE BOARD THROUGH N MOVES
 40 FOR I = 1 TO N
 45 REM GENERATE A RANDOM MOVE LIKE ROLLING DICE
 50 LET M = RND(6) + RND(6)
 55 REM FIND NEW POSITION BY ADDING MOVE TO OLD POSITION
 60 LET P = P + M
 65 REM CHECK IF PLAYER HAS REACHED OR PASSED GO
 70 IF P < 41 THEN 90
 75 REM IF SO, THEN PIECE HAS MOVED AROUND THE BOARD
 77 REM AND WE MUST ADJUST ITS POSITION APPROPRIATELY
 80 LET P = P - 40
 90 PRINT "ROUND NUMBER", I, "POSITION", P
100 NEXT I
999 END
```

Note that RND(6) generates a different random number each time, so RND(6) + RND(6) is not necessarily equal to either twice RND(6) or to RND(12).

BRIEF REVIEW

We have introduced the statement DIM and the random number generator RND. Remember the following:

- —You can refer to an element of a set by means of the name of the set and the element number. For example, A(6) is the sixth element of the set A.
- —You must specify the maximum number of elements in each set with a DIM statement. The simplest approach is to enter the DIM statements after you enter the rest of the program.
- —RND produces a number that is equally likely to have any of the values in its range. Each use of RND results in a different random number, much as does each roll of an honest die or toss of an honest coin.

CONCLUSION

We have not described every feature of the BASIC language. Rather, we have shown how you can tell the computer to perform tasks and how programs implement some of the things you have probably seen computers do. Although we talk about computers as if they were intelligent, the real intelligence is in the programs that tell the computers what to do.

REFERENCES

Albrecht, R. *My Computer Likes Me.* Menlo Park, CA: People's Computer Co., 1975.
Albrecht, Finkel, and Brown. *BASIC for Home Computers.* New York: Wiley, 1978.
Brown, J. *Instant BASIC.* Menlo Park, CA: Dymax, 1977.
Coan, J. S. *Advanced BASIC.* Rochelle Park, NJ: Hayden, 1977.
Coan, J. S. *Basic BASIC.* Rochelle Park, NJ: Hayden, 1970.
Kemeny, J., and Kurtz, I. *BASIC Programming.* 2d ed. New York: Wiley, 1971.
Lien, D. *The BASIC Handbook.* San Diego, CA: CompuSoft Publishing, 1979.
Poole, L., and Borchers, M. *Payroll with Cost Accounting in BASIC.* Berkeley, CA: Osborne/McGraw-Hill, 1977.
Poole, L., and Borchers, M. *Some Common BASIC Programs.* Berkeley, CA: Osborne/McGraw-Hill, 1977.
Spencer, D. *Game Playing with Computers.* Rochelle Park, NJ: Hayden, 1975.
"What to Do After You Hit Return." Menlo Park, CA: People's Computer Co., 1975.

Note: Heath Company, Benton Harbor, MI 49022, offers a complete continuing education course entitled *BASIC Programming* (Model EC-1100), which includes a self-instruction manual, workbooks, and examinations.

4 How To Write Programs

In Chapter 3, we discussed the writing of programs in BASIC and presented many short examples, each illustrating a particular feature of the language. This is like explaining how to direct a play by showing the student separately how to arrange an entrance, an exit, a cue, a monologue, a dialog, and a crowd scene. Real plays, of course, involve all these things and the director must fit them together. In the same way, real programs use many features of a computer language, and the programmer must fit them together.

In this chapter, we discuss the general methods and problems involved in writing realistic programs. Some of the questions with which we deal are

- —How do you describe what a program should do? Where do you start?
- —How do you outline and implement a program so you can make it work correctly in a reasonable amount of time?
- —What do you do if the program doesn't work? Where are the errors likely to be and how can you find and correct them?
- —How do you ensure that the program really does what you want?
- —How do you document the program so you can use it later?

These questions are important even if you buy programs or pay someone to write them, You still must specify what the program should do, decide how to test it, and learn how to use it. Therefore, you need some understanding of how programs are defined, coded, debugged, tested, and documented. With this understanding, you can describe what you want, evaluate what's available, estimate how long a program should take to write and how much it should cost, and determine what you need to use and maintain a program. You need not be an expert programmer to be a knowledgable user, capable of explaining your needs, evaluating products, and recognizing misrepresentations and incomplete descriptions.

Of course, the questions we have asked do not have simple answers. Writing programs is like writing books, composing music, or painting pictures—there are few absolute rules. Rather, there are styles and schools of thought. We describe some guidelines that come from programming experience and some methods that are widely used. You should become aware of the problems involved in writing programs, even if you never write any yourself.

THE STAGES OF PROGRAM WRITING

To write a program, you must go through the following stages:

Problem definition, determining what the program must do
Program design, outlining the procedure the computer must follow to do the job
Coding, translating the design into an actual program, a series of instructions in a form the computer can understand
Debugging, discovering and correcting errors (or "bugs") in the program
Testing, determining whether the program does what you want. This is the "quality control" stage
Documentation, describing the program so it can easily be used, maintained (improved and corrected), and extended (We discuss the maintenance of programs in Chapter 7.)

We should note some interesting findings about time and money based on the experience of those who develop large commercial programs. First, the time factor: Project managers typically allocate 40% of the programming time to problem definition and program design, 20% to coding, and 40% to debugging, testing, and documentation. An often used rule-of-thumb is that a programmer can write one to ten fully debugged and documented lines of program per day. This amount does not depend on the language used, but you should remember that programs written in high-level languages generally contain far fewer lines than programs written in assembly language. Thus most programmers spend relatively little of their time writing instructions (coding); they spend most of their time defining problems, outlining programs, checking programs, and writing documentation.

Second, the money factor: The cost of writing a program may be more than $50 per line. Furthermore, this cost is rising even though the cost of computers is decreasing. Large programming projects often exceed initial estimates of cost and time by 50% to 200%. There are few reliable ways to estimate how long it will take someone to write a program. The most difficult stages to figure are debugging and testing, since most programmers underestimate how long it will take to find all the errors in a program and check it out thoroughly.

A major problem is lack of standards. Much of programming is an art that you can learn only from experience. There are few rules for the newcomer to follow, particularly in the time-consuming stages of definition, design, debugging, testing, and documentation. The lack of standards affects those who use and buy the programs as well as those who write them, since there is no standard way to define what a program does, describe how it was tested, or explain how to use it. Users often have to develop their own standards.

Programming is highly labor-intensive, so the cost of writing programs increases with inflation. Computers may be getting faster, smaller, and cheaper, but neither programmers nor programming methods are improving at a comparable rate. Factories can produce powerful computers on tiny chips for a few dollars apiece, but the use of those computers still depends on a programmer's ability to define tasks, write instructions, locate and correct errors, and test and document programs.

PROBLEM DEFINITION—THE STARTING POINT

The first question to ask is "What should the program do?" This is an obvious question, but one that surprisingly often never gets answered. In writing programs, as in other projects, it's tempting to start solving the problem before sitting down and determining what the problem is.

How do you define the problem? Some factors to be considered are

Inputs
Outputs
Processing requirements
Error handling
Peripherals
Other programs
Operating methods
Design and development methods
Long-range needs

Let us look at these factors in more detail. The definition of *inputs* and *outputs* must include: rate, type, range of values, format, order, and medium.

The definition of *processing requirements* must include: calculations to be performed on each type of data, accuracy, order of operations, time constraints, and memory constraints.

The definition of *error handling* must describe the possible errors and most likely errors, the errors that should be recognized and recorded, and how each type of error is to be handled.

The *other programs* may include the operating system, other systems programs, and other applications programs. *Operating methods* may include the startup procedure, the way in which data will be entered, and the way in which the program will interact with the operator. *Design and development methods* refer to the design techniques, coding methods, debugging plan, test plan, and documentation standards. *Long-range needs* include expansion to larger problems, extension to new or more complex tasks, portability (i.e., moving the program to another computer), and addition of new peripherals or systems software.

The definition stage usually requires some study. You may have to learn how your computer handles inputs and outputs or how its operating system works. You may want to check how the peripherals work and how you can use the programs you already have. The key point is to investigate things early, before you are committed to a course of action. The problem definition stage is the time to identify potential problems when you can still solve them easily and cheaply.

Example: A Payroll Program

If, for example, you want to write a payroll program, you must define the following:

INPUTS
Number

Type
Format
Order
Medium

PROCESSING REQUIREMENTS
Equations that relate results to input data
Accuracy needed
Time and memory constraints

OUTPUTS
Number
Type
Format
Order
Medium

ERROR HANDLING
Data errors

OPERATING METHODS
Startup
Data entry
Interaction with the operator

OTHER PROGRAMS
Operating system
Accounting programs

If the company has twenty employees with five pay categories and overtime, the information might be as follows:

Inputs

Keyboard entry: Each entry consists of employee number, number of regular hours worked, and number of overtime hours worked. The final entry has employee number zero.

Tape record file: Each record consists of employee number, date of hire, date of termination, pay category, federal tax deduction rate, state tax deduction rate, number of regular hours worked year-to-date, number of overtime hours worked year-to-date, total salary paid year-to-date, total federal tax deductions year-to-date, total state tax deductions year-to-date, and total other deductions year-to-date.

Processing

An equation calculates net pay from number of hours worked, number of overtime hours worked, pay category, federal tax deductions, state tax deductions, and other deductions. All yearly totals in the record must be updated.

Outputs

Each employee must receive a check for his or her net pay and a statement reporting net pay, gross pay, deductions, hours worked, and year-to-date totals.

Error Handling

Possible errors include an excessive number of hours, nonexistent or terminated employee, and total deductions exceeding gross pay. The program should recognize any of these errors immediately and report them to the operator.

Operating methods

The operator can start the program from the operating system by typing "LOAD PAYROLL". The program prompts the operator for each entry, explaining what must be entered and in which order. The operator is also told that entering employee number zero terminates the program. The operator can request a summary of payroll information before the program returns control to the operating system.

Among the questions that you must answer during problem definition are: How do you enter a new employee into the records? How do you remove a terminated employee from the records? How do you determine, change, and update pay categories and deductions? How many employees and how many categories must the program be able to handle? Can the program handle deductions that only apply up to a certain total salary, such as social security taxes or state disability insurance? Will the program print checks directly or merely provide the amounts? Here the availability of peripherals is important. Is the program compatible with other programs, such as general accounting, mailing lists, or personnel records? How will you maintain security so unauthorized people cannot examine or change the records? How will errors be reported and corrected? What should the program regard as an excessive number of hours? How can duplicate entries (and duplicate checks) be eliminated?

Obviously you must answer such questions, regardless of whether you are going to write the program yourself, purchase it off-the-shelf, or pay someone to write it.

A clear and thorough problem definition is vital in all subsequent stages of program writing. It is ridiculous to design, code, debug, and test a program when you do not know what it is supposed to do. Time spent in problem definition can help you avoid a great deal of frustration and wasted effort.

PROGRAM DESIGN

The next stage of program writing is *design*. Now that we know the requirements, we must outline the program that will meet them. Program design is like doing blueprints for a building, the drawings for an engine, or the schematic for a television set. Among the methods commonly used to design

programs are flowcharting, top-down design, modular programming, and structured programming.

Let us first decide what we want the outline to provide. We would like it to furnish the following:

A clear picture of how the program works. We want to see at a glance what the program does and how it handles different types of data.

A description from which we can easily derive the actual program. This requires a standard set of forms that we can translate automatically into statements in a computer language.

A way to recognize errors and problems, so that we can correct them before we write the program. The earlier we find errors, the easier and cheaper they are to correct.

A way to divide a large program into manageable pieces that can be coded separately, checked out, and combined.

Documentation that explains how the program works to other programmers and to users.

A blueprint does this for a building and a drawing or schematic does it for a piece of mechanical or electrical equipment.

Flowcharting

A diagram is often the best way to describe how things work. As the adage goes, "a picture is worth a thousand words." A flowchart like the one in Fig. 4-1 lets the reader see the order of operations, the decision points, and the relationships between the sections of the program.

Flowcharting is popular enough that a set of standard symbols has been developed; Figure 4-2 shows the most widely used of these, There are templates for drawing flowcharts and computer programs that will produce them. An advantage of flowcharts is that even people with little computer background can understand them.

Example 1: A Simple Decision

A simple example is a program that compares a student's answer to the correct answer. If the answers are the same, the program adds one to the student's score. Figure 4-3 contains a flowchart of the program, showing clearly the order of operations and the way in which the computer determines whether the answer is correct.

Example 2: Social Security Deductions

A more complex program is one that calculates social security deductions for a payroll. The amount to be deducted for social security is a specified percentage of all wages under a ceiling (or maximum income). The variables are total wages year-to-date, gross wages for the current pay period, percentage owed for social security tax, and the maximum income subject to social security tax.

The program must produce the following results: (1) If the total wages already exceed the ceiling, the deduction is zero. (2) If total wages plus the wages for this pay period exceed the ceiling, the deduction is a percentage of the difference be-

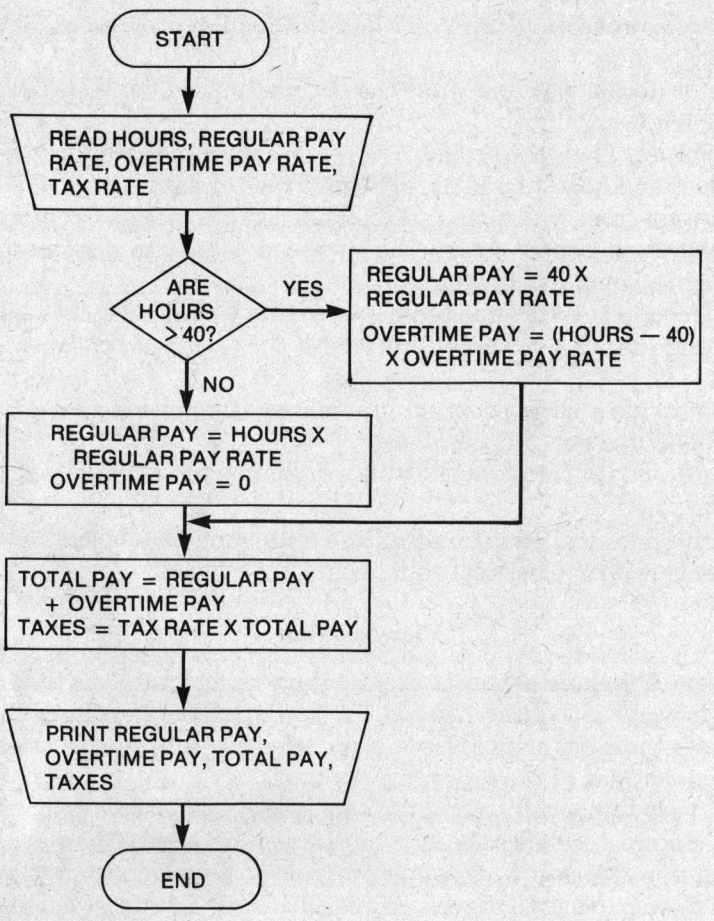

START
READ HOURS, REGULAR PAY RATE, OVERTIME PAY RATE, TAX RATE
ARE HOURS > 40?
YES
REGULAR PAY = 40 X REGULAR PAY RATE
OVERTIME PAY = (HOURS — 40) X OVERTIME PAY RATE
NO
REGULAR PAY = HOURS X REGULAR PAY RATE
OVERTIME PAY = 0
TOTAL PAY = REGULAR PAY + OVERTIME PAY
TAXES = TAX RATE X TOTAL PAY
PRINT REGULAR PAY, OVERTIME PAY, TOTAL PAY, TAXES
END

Figure 4-1. A typical flowchart

tween the ceiling and the total wages. (3) Otherwise, the deduction is a percentage of the wages for this pay period.

Figure 4-4 is the flowchart of this procedure. Here there are two decision points and five operations. Clearly flowcharts become more difficult to draw and to understand as the program becomes more complicated.

There are some ways to deal with the problem of complex flowcharts. Some programmers write two flowcharts, one that includes only the key points and one that has more details. The one with the key points provides an overview of the program and guidelines for understanding the more detailed flowchart.

Another approach is to draw two sets of flowcharts: one (called *program flowcharts*) follows the program and the other (called *data flowcharts*) follows particular kinds of data through the program. In the payroll example a program flowchart would show startup, data input, calculations, and printing of results. A set of data flowcharts would show separately how the program processes permanent employees, temporary employees (see Figure 4-5), regular pay, overtime pay, and special deductions. Thus a data flowchart shows one path through the

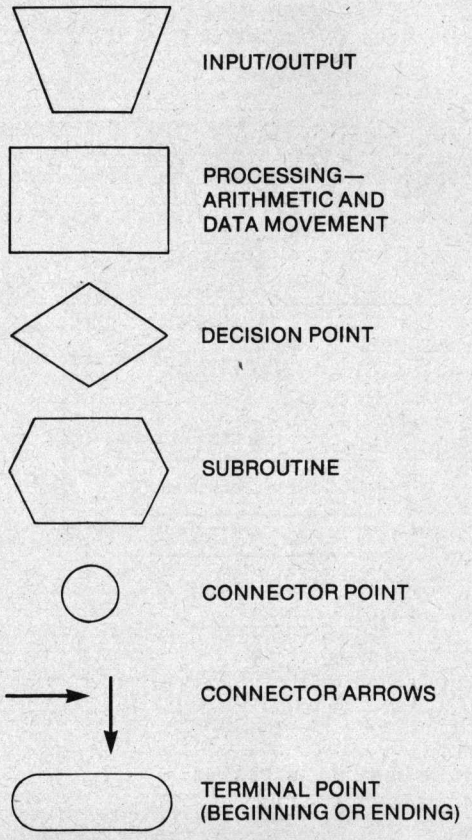


Figure 4-2. Standard flowchart symbols

program, much as travel directions show one path through the highway system. Data flowcharts are often handy. If, for example, our payroll program was handling temporary employees incorrectly, we could look at the data flowchart in Figure 4-5 and immediately see which parts of the program were involved. If we had only the program flowchart, we would have to search through it, much like finding our way home from a road map.

We recommend the following general approach to flowcharting:

(1) Draw a rough flowchart to get yourself started. Don't make it too detailed or artistic.
(2) Update the flowchart as you design and code the program.
(3) Don't let the flowchart occupy too much of your time. If you get bogged down in drawing connections and arranging symbols, try another design method.
(4) Include a flowchart in the final documentation. This flowchart should provide an overview of the program.

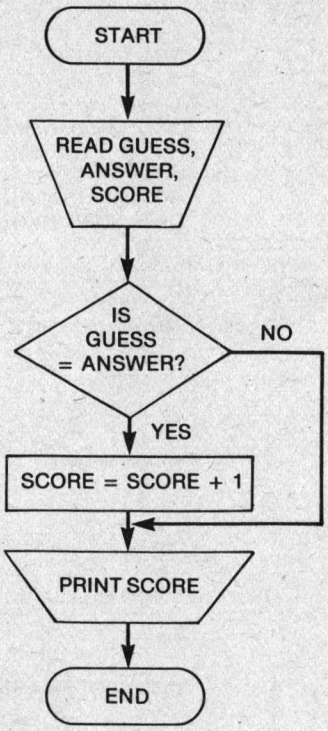


Figure 4-3. Flowchart for checking an answer

A DESIGN PROCEDURE

Once you have drawn a flowchart, we strongly recommend that you proceed as follows:

Start with the overall program and gradually increase the amount of detail. This approach is called *top-down design.*

Proceed in small steps, dividing each task at each level of detail into small sections or *modules* that interact as little as possible. This method is called *modular programming.*

Describe each module using a small number of standard forms. Make the description proceed in a straight line (no backtracking or jumping ahead). This method is called *structured programming.*

Top-down design, modular programming, and structured programming together form a complete design procedure. Top-down design tells you where to start, modular programming tells you how to divide the overall job into reasonable steps, and structured programming tells you how to describe these individual steps.

TOP-DOWN DESIGN

Programmers must divide long programs into a set of manageable short programs. The problems are how to divide the long programs and how to put the

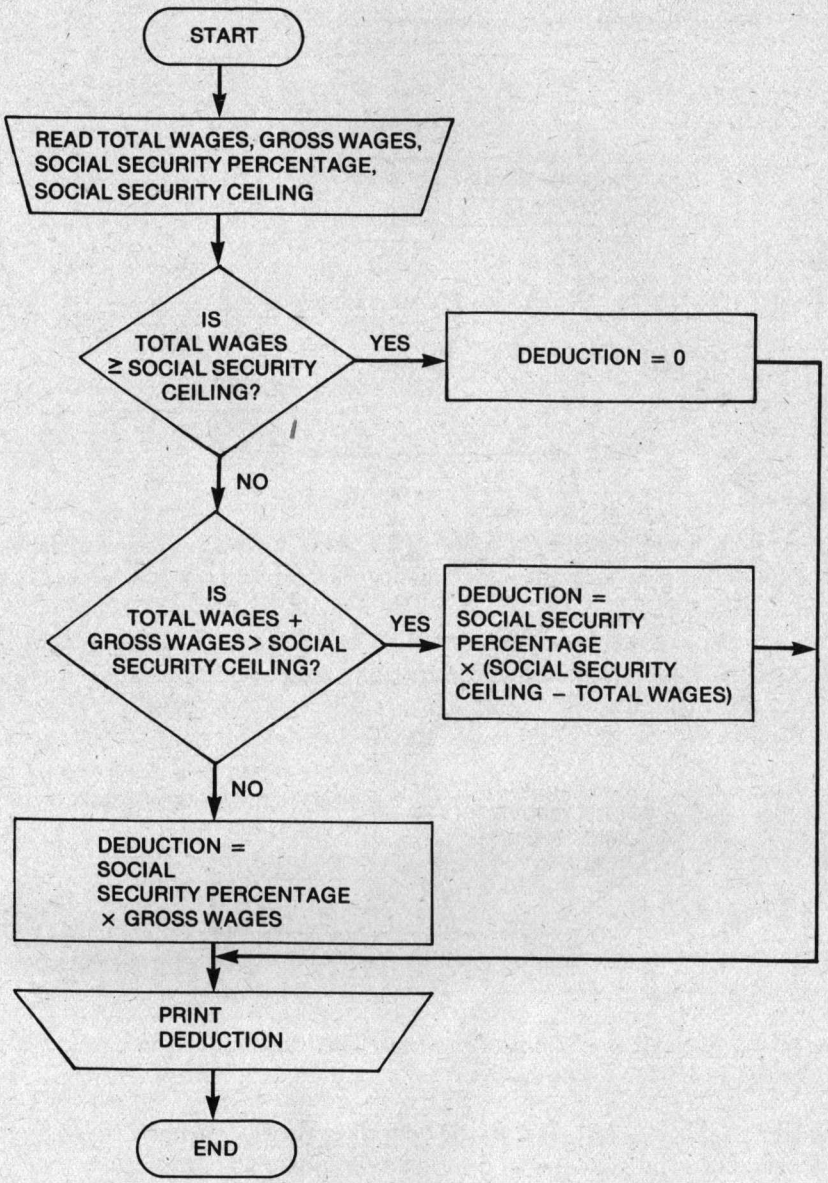

START
READ TOTAL WAGES, GROSS WAGES, SOCIAL SECURITY PERCENTAGE, SOCIAL SECURITY CEILING
IS TOTAL WAGES ≥ SOCIAL SECURITY CEILING?
YES
DEDUCTION = 0
NO
IS TOTAL WAGES + GROSS WAGES > SOCIAL SECURITY CEILING?
YES
DEDUCTION = SOCIAL SECURITY PERCENTAGE × (SOCIAL SECURITY CEILING − TOTAL WAGES)
NO
DEDUCTION = SOCIAL SECURITY PERCENTAGE × GROSS WAGES
PRINT DEDUCTION
END

Figure 4-4. Flowchart for computing social security deductions

pieces together. In the top-down design method illustrated in Figure 4-6, the idea is to build the overall framework first (like starting a jigsaw puzzle from the outside frame or a building from its outer shell) and then gradually to fill in the details. *Stubs* (or proxies) are used in place of programs that have not yet been written. A stub is a fill-in, used like rough boards of the proper size that are often substituted for doors or windows during construction of a building.

In top-down design, each new piece goes in its intended place, rather than sitting off by itself. Thus the programmer adds each piece to a framework, instead of producing many separate pieces that must be combined later. The program-

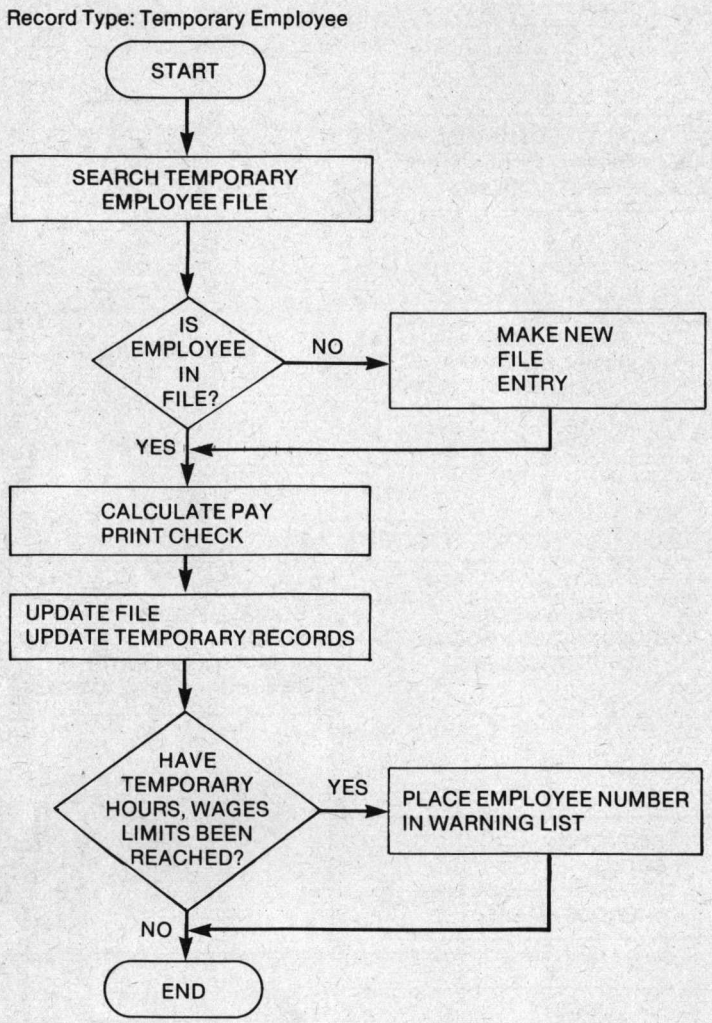


Figure 4-5. An example of a data flowchart. This flowchart shows how a large program handles a particular type of data.

mer finishes a piece, inserts it into the overall program, checks the program, and then proceeds to the next piece, moving ever closer to completion.

To illustrate the use of top-down design in writing BASIC programs, we need a way to tell the computer to execute an entire program rather than a single statement. Initially the programs (often called *subroutines* since they are subordinate to the main program) will be the stubs we mentioned. Later we will expand them without changing the main program. The two new BASIC statements that we need are:

GOSUB (execute a subroutine starting at the specified line number)
RETURN (go back to the main program, starting at the line immediately after the one that transferred control to the subroutine)

Thus the statement GOSUB 2000 tells the computer to execute the subroutine that starts at line 2000. A RETURN at the end of the subroutine tells the computer to resume the main program. We usually place subroutines at high line numbers so they do not interfere with the main program.

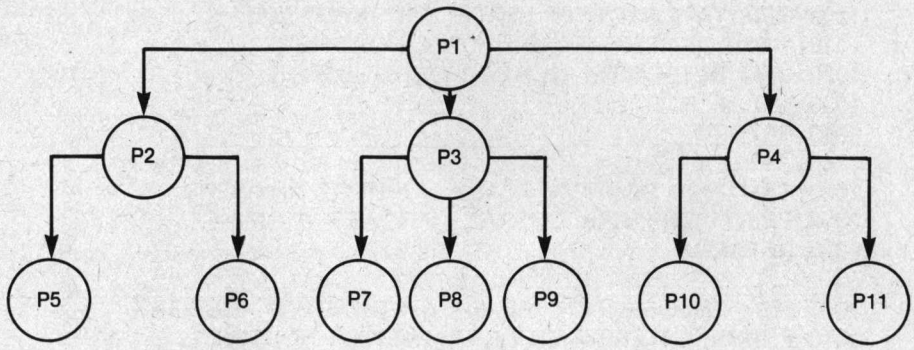


Figure 4-6. The top-down design procedure

Example: A Test Scorer

The task is to write a program that will grade a set of multiple-choice tests. The teacher must enter the number of students, the number of questions, and the correct answers. Then the teacher must enter each student's answers; the computer will grade each test and print the score. As an extra feature, the teacher can enter the smallest and largest valid answers; the computer will check the actual answers to see if they are within the specified limits. This procedure (called *limit checking*) warns the teacher about input errors (e.g., entering 8 as an answer when there are only five choices).

To begin we divide the program into the following sections:

```
READ INPUT DATA
READ CORRECT ANSWERS
READ STUDENT ANSWERS
GRADE EXAM
PRINT RESULTS
```

Figure 4-7 is a flowchart of the overall program.

Our initial BASIC program does little more than direct the computer to one subroutine after another. The main program is complete; it will remain virtually the same after we expand the stubs. The initial subroutines (or stubs) just print messages, except for the first subroutine, which sets the number of students to 1 for testing purposes.

```
 10 REM READ THE INPUT DATA
 20 GOSUB 1000
 30 REM READ THE CORRECT ANSWERS
 40 GOSUB 2000
 50 REM GRADE EXAMS AND PRINT SCORES
 60 FOR I = 1 TO N
 70 REM READ STUDENT'S ANSWERS
 80 GOSUB 3000
 90 REM GRADE THE EXAM
100 GOSUB 4000
110 REM PRINT THE SCORE
120 GOSUB 5000
130 NEXT I
999 END
```

```
1000 REM THIS SECTION READS THE INPUT DATA
1010 PRINT "ENTERED INPUT DATA ROUTINE"
1020 REM DETERMINE NUMBER OF STUDENTS
1030 LET N = 1
1040 RETURN

2000 REM THIS SECTION READS CORRECT ANSWERS
2010 PRINT "ENTERED CORRECT ANSWER ROUTINE"
2020 RETURN

3000 REM THIS SECTION READS A STUDENT'S ANSWERS
3010 PRINT "ENTERED STUDENT ANSWER ROUTINE"
3020 RETURN

4000 REM THIS SECTION GRADES THE EXAM
4010 PRINT "ENTERED EXAM GRADING ROUTINE"
4020 RETURN

5000 REM THIS SECTION PRINTS THE STUDENT'S SCORE
5010 PRINT "ENTERED SCORE PRINTING ROUTINE"
5020 RETURN
```

Running this program shows that the overall approach is correct. We do not have to enter any data at all. The program simply executes each stub, printing a message to inform us of its progress. The printed results are

```
ENTERED INPUT DATA ROUTINE
ENTERED CORRECT ANSWER ROUTINE
ENTERED STUDENT ANSWER ROUTINE
ENTERED EXAM GRADING ROUTINE
ENTERED SCORE PRINTING ROUTINE
```

Now let us expand the first two stubs. We will make the first stub ask the operator to enter the number of students N. We will make the second stub ask the operator to enter the number of questions Q and the correct answer for each question. The rest of the program will remain as it was.

```
1000 REM THIS SECTION READS THE INPUT DATA
1010 REM ASK OPERATOR TO ENTER NUMBER OF STUDENTS
1020 PRINT "HOW MANY STUDENTS"
1030 INPUT N
1040 RETURN

2000 REM THIS SECTION READS CORRECT ANSWERS
2010 REM ASK OPERATOR TO ENTER NUMBER OF QUESTIONS
2020 PRINT "HOW MANY QUESTIONS"
2030 INPUT Q
2040 REM ASK OPERATOR TO ENTER CORRECT ANSWERS
2050 FOR J = 1 TO Q
2060 PRINT "ENTER CORRECT ANSWER TO PROBLEM", J
2070 INPUT C(J)
2080 NEXT J
2090 RETURN
```

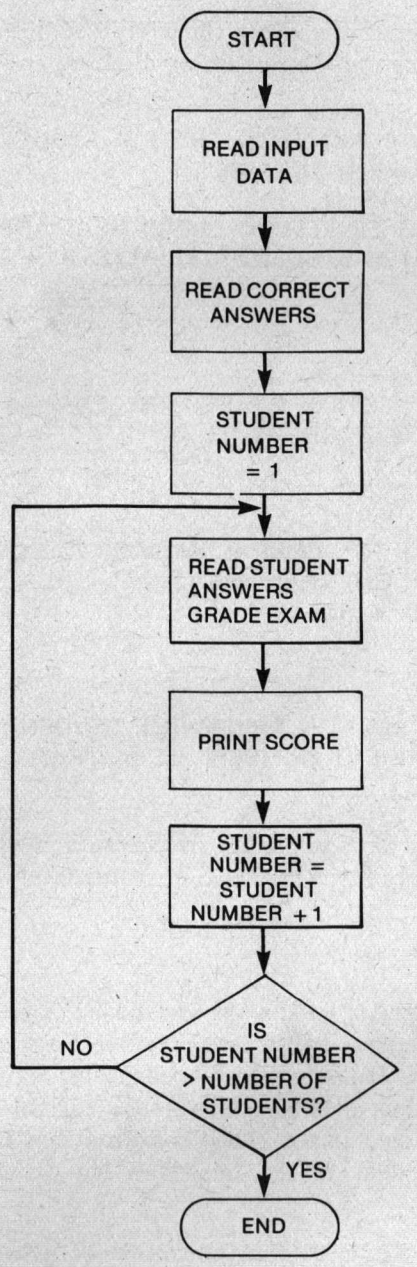


Figure 4.7. Flowchart for the initial test scoring program

We also have to add a DIM statement to the main program (see Chapter 3), since we have introduced a set C (the correct answers). The added statement is

```
5 DIM C(25)
```

We can now check this program for a simple case. For example, we could assume that there is only one student and only one question; this test should reveal any serious errors in the program. Note how we left most of the program

alone and fit the new statements into the overall structure.

We can proceed to expand the remaining modules as follows:

```
3000 REM THIS SECTION READS A STUDENT'S ANSWERS
3010 REM ASK OPERATOR TO ENTER A STUDENT'S ANSWERS
3020 PRINT "STUDENT NUMBER", I
3030 FOR J = 1 TO Q
3040 REM ASK OPERATOR TO ENTER NEXT ANSWER
3050 PRINT "ENTER STUDENT'S ANSWER TO PROBLEM", J
3060 INPUT G(J)
3070 NEXT J
3080 RETURN

4000 REM THIS SECTION GRADES THE EXAM
4010 REM INITATE SCORE (S) TO 0
4020 LET S = 0
4030 REM GRADE THE EXAM
4040 FOR J = 1 TO Q
4050 REM ADD 1 TO SCORE IF ANSWER IS CORRECT
4060 IF C(J) <> G(J) THEN 4080
4070 LET S = S + 1
4080 NEXT J
4090 RETURN

5000 REM THIS SECTION PRINTS THE STUDENT'S SCORE
5010 PRINT "STUDENT NUMBER", I, "SCORE", S
5020 RETURN
```

Here again, all that we have to change in the main program is the DIM statement (since we have introduced the set G of student answers). The revised DIM statement is

```
5 DIM C(25), G(25)
```

We can add more details to the subroutines. For example, let us ask the operator to enter the largest and smallest possible answers. The program will then reject answers that are not within those limits. Figure 4-8 is a flowchart of the limit checking routine; all that it does is set a variable M to 0 if the answer is within the limits and to 1 if not. The programs that use the limit checking routine must decide what to do, based on the value of M. The revised subroutines are

```
2000 REM THIS SECTION READS CORRECT ANSWERS
2010 REM ASK OPERATOR TO ENTER NUMBER OF QUESTIONS
2020 PRINT "HOW MANY QUESTIONS"
2030 INPUT Q
2032 REM ASK OPERATOR TO ENTER SMALLEST AND LARGEST
2033 REM       VALID ANSWERS
2034 PRINT "ENTER SMALLEST, LARGEST VALID ANSWERS"
2036 INPUT L, H
2040 REM ASK OPERATOR TO ENTER CORRECT ANSWERS
2050 FOR J = 1 TO Q
2060 PRINT "ENTER CORRECT ANSWER TO PROBLEM", J
```

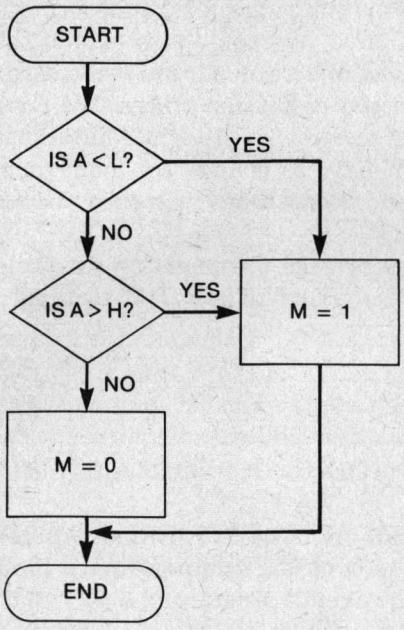


Figure 4-8. Flowchart for limit checking subroutine

```
2070 INPUT A
2080 REM CHECK IF ANSWER IS WITHIN THE LIMITS (M = 0)
2090 GOSUB 2500
2100 IF M = 0 THEN 2150
2110 REM ANSWER IS OUTSIDE LIMITS, INFORM OPERATOR
2115 REM       OF ERROR
2120 PRINT "ANSWER NOT WITHIN LIMITS — ENTER IT AGAIN"
2130 GOTO 2060
2140 REM ANSWER IS VALID, SO SAVE IT IN THE SET
2150 LET C(J) = A
2160 NEXT J
2170 RETURN

2500 REM THIS SECTION DETERMINES IF AN ANSWER IS VALID
2510 REM ANSWER VALID (M = 0) IF A IS BETWEEN L AND H
2520 REM ANSWER INVALID (M = 1) IF A OUTSIDE LIMITS
2530 REM INVALID IF BELOW L
2540 IF A < L THEN 2610
2550 REM INVALID IF ABOVE H
2560 IF A > H THEN 2610
2570 REM IF WE GOT HERE, A IS WITHIN LIMITS, M = 0
2580 LET M = 0
2590 RETURN
2600 REM IF WE GOT HERE, A IS OUTSIDE LIMITS, M = 1
2610 LET M = 1
2620 RETURN
```

We could use the same limit checking subroutine to check the student's answers as they are entered. We could also extend the grading routine so it keeps track of which questions a student answered correctly or gives full credit for one answer and partial credit for others. We could extend the printing routine so it records the score in the student's file and assigns it a letter grade. In each case, we do not have to change the main program; all that we must change is one routine and it continues to occupy the same place in the overall structure.

Each level of detail serves as a foundation for the next level. We did not change the main program except to add DIM statements as we moved from a bare skeleton to a useful program.

MODULAR PROGRAMMING

Top-down design depends on the idea of dividing a long program into short subprograms. We usually refer to the subprograms as *modules*.

We want to divide the overall program in a way that achieves the following goals:

The modules should perform tasks that we are likely to need in many programs. This will save us work in the long run.

We should be able to use programs that we already have. These may be modules that we wrote earlier, programs that we have purchased, or routines that are part of our systems software.

We should be able to make likely changes by modifying a single module. Likely changes might include new input or output peripherals, new methods for processing data or performing calculations, new input or output formats, increased interaction with the operator, and more error checking.

The modules should act independently. The idea here is for us to be able to test or change one module without needing the others or affecting them. The program should be organized like the spy networks in popular movies; each person in the spy network knows as little as possible about the other people so that anyone can be removed or replaced without affecting the flow of information. The idea that each module should know as little as possible about other modules is often referred to as Parnas's "information-hiding principle".

Dividing programs into modules is difficult. We suggest the following guidelines:

Keep the modules short, not more than 20 or 30 lines long.

Be general. Decide what factors you might want to vary. A module that performs one highly specific task, such as ordering a set of seven elements, is unlikely to be useful again.

Make the inputs straightforward and the outputs clear and immediately useful. Preparing the data and interpreting the results should be as simple as possible.

Implement specific procedures or features in a single module. You should be able to change them by modifying one module, rather than several.

Use subroutines to limit the effects of changes. You can revise the subroutines without affecting the main program.

Reorganize the modules if you find that you divided the program incorrectly. A good set of modules will be easy to understand and use again.

Design each module for a single job. Don't combine jobs that you might want to separate later.

Example: Test Scoring

The grading program that we discussed as an example of top-down design is also a typical example of modular programming. The modules perform the following tasks.

- —Reading the overall input data: This module could handle interactive or non-interactive input, different data formats, and different ways of classifying or identifying students, such as by section or grade level.
- —Reading the correct answers: This module could handle interactive or non-interactive input, different input devices, different data formats, and different amounts of error checking.
- —Reading the student's answers: This module could handle a situation in which we obtained the correct answers from one input device (e.g., a cassette recorder) and the student answers from another device (e.g., a mark-sense card reader).
- —Scoring the examination: This module could handle partial credit, identify particular questions that were answered correctly or incorrectly, or arrange the questions by level of difficulty or by subject area.
- —Printing the results: This module could handle different output devices, formats, or choices of output data.

Still other modules could print headings, provide a summary at the end, order the scores numerically, determine the average, and assign letter grades.

Obviously, these modules cannot be completely independent of each other since many of them use the same data. But most of the choices are implemented in a single module. We could change input devices, output devices, grading policies, data formats, and error checking methods by changing a single module. We would not have to worry about introducing errors in completely unrelated places.

STRUCTURED PROGRAMMING

The question remains how to describe a module. We need a method that is easy to understand, allows us to show loops and decisions with a few simple forms, and produces a description that is straightforward and easy to follow. Structured programming comes close to meeting all these needs. This method is based on the mathematically proven fact that any program can be described using only three forms or structures:

- —A *consecutive form* in which the computer executes parts of the program one after another, moving down the page
- —A *conditional form* that lets the computer decide whether to execute part of the program
- —A *loop form* that lets the computer repeat part of the program

The consecutive form simply defines the computer's normal behavior, so we only have to discuss the other two forms (the decision and the loop). The most popular conditional form (see Figure 4-9 for a flowchart) is the IF-THEN-ELSE statement in which a condition follows IF, one alternative program follows THEN, and the other follows ELSE. Some examples are

Example: *If* Y ≠ 0 *then* X = 1/Y *else* X = 0

The result is

$$X = \begin{cases} 0 & \text{if } Y = 0 \\ 1/Y & \text{otherwise} \end{cases}$$

This statement describes a procedure whereby the computer avoids dividing by zero.

Example: *If* CENTS ≥ 50 *then* DOLLARS = DOLLARS + 1

The result is

$$\text{DOLLARS} = \begin{cases} \text{DOLLARS} + 1 \text{ if CENTS} \geq 50 \\ \text{DOLLARS if CENTS} < 50 \end{cases}$$

This statement describes a procedure for rounding to the nearest dollar. Here we omitted ELSE, since no action is necessary if less than fifty cents is involved.

Example: *If* VALUE > THRESHOLD *then* PASSED *else* FAILED

Here PASSED and FAILED could be entire programs which might report the results, request more information from the user, or perform other tasks. An IF-THEN-ELSE statement is like the choice between travelling on a regular highway and using a bypass: Both start at the same place and take you to the same final destination.

The most popular loop form (see Figure 4-10 for a flowchart) is the DO-WHILE statement in which the computer executes a program as long as a condition following WHILE is true. Of course executing the program must eventually make the condition false or else the computer will repeat the program forever.

Here are some DO-WHILE statements.

Example: Form a sum by adding together five elements from the set A, starting with A(1); that is, SUM = A(1) + A(2) + A(3) + A(4) + A(5). Remember that A(I) is the Ith element of the set A.

```
SUM = 0
I = 1
do while I≤5
    SUM = SUM + A (I)
    I = I + 1
    end
```

The "*end*" statement marks the boundary of the program controlled by the DO-WHILE statement. The program includes the statement I = I + 1, which

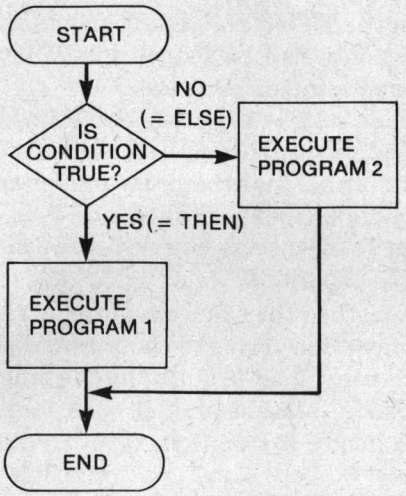


Figure 4.9. Flowchart for an IF-THEN-ELSE statement (if CONDITION then PROGRAM 1 else PROGRAM 2)

will eventually make the loop terminate. DO-WHILE statements are generalizations of the BASIC FOR-NEXT statements introduced in Chapter 3.

Example: Subtract 60 from any of the five elements in the set M that are 60 or larger. Here M could represent the number of minutes it took a student to complete a test. Of course, we also have to add 1 to the corresponding element of the set H (hours) if we subtract 60 from an element of M.

```
I = 1
do while I≤5
    if M(I)≥60 then
      M(I) = M(I) − 60
      H(I) = H(I) + 1
    I = I + 1
    end
```

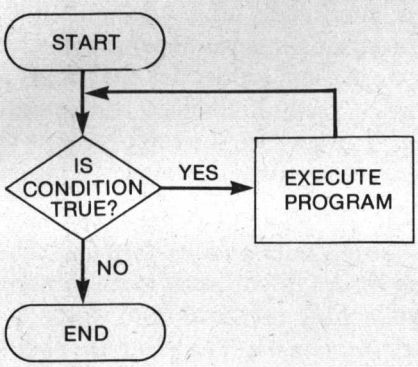


Figure 4-10. Flowchart for a DO-WHILE statement (do PROGRAM while CONDITION)

This example includes both a DO-WHILE statement and an IF-THEN-ELSE statement. These statements can be lodged inside one another indefinitely (called *nesting*) to describe complex programs.

Structured programming refers to any design method in which programmers use only conditional forms and loop forms, besides the normal consecutive form. There are many possible forms and the programmer can use several of each type, much as a highway department can choose different sets of signs to direct drivers. The forms we have discussed, however, are a small set that is convenient for describing simple programs.

Structured programming has the same advantages as using any set of standard forms. There are only a few forms to choose from, so you have only a few alternatives to consider at any point in outlining the program. Each decision requires an IF-THEN-ELSE statement and each loop a DO-WHILE. The limited number of forms makes descriptions repetitive and thus easy to understand, change, and use.

Furthermore, the description produced using structured programming always proceeds straight ahead. It never says "Go back to step 7" or "Jump to step 23". This makes the description easy to follow and also means that you always know how you reached a particular point. Remember, each form has a single entry and a single exit, so you can retrace your steps and change part of the description without worrying about unpredictable side effects.

Of course, after we have used structured programming, we still have to translate the description into an actual computer program. It would clearly be helpful if the computer would do this for us. Unfortunately, older computer languages such as BASIC and FORTRAN were invented before structured programming and do not include the required forms. PASCAL, on the other hand, is a language invented specifically for structured programming; this is a major reason for its increasing popularity. If we write our description in PASCAL, it can serve as an actual computer program.

CODING

Coding, the next stage of programming, is simply the writing of instructions in a form that the computer can understand and eventually execute. The specific form depends on your computer, language, and application. Many books describe how to write programs in specific languages for particular computers and applications. Here we present guidelines that apply in most situations. The original source of many of these guidelines is Kernighan and Plauger's *The Elements of Programming Style* (see the references at the end of this chapter).

All the following guidelines emphasize defensive programming, i.e., coding to minimize the likelihood of errors.

Keep the program simple and straightforward. Programs are confusing enough without inviting errors. If you want to do something tricky, take a short break and you'll probably think better of it.

Make changes easy to implement. The program should be organized so that

changes are not a major hassle. You will surely make some changes so you might as well be prepared for them.

Avoid the obvious confusions. A program should not contain names that are too similar or other things that lead to confusion. Clear-cut distinctions are always helpful, particularly when a tired programmer is trying to find that last baffling error.

Stay on firm ground. Remember, you don't have to use every feature of your computer unless you're trying to get into the Guinness Book of World Records. Use the features that you understand. Try something new once in a while but be cautious.

Make the program readable. A readable program will be easy for people to understand.

Be repetitive. Use the same approach, the same data formats, and the same methods whenever possible, even if it is a bit wasteful. Making everything different leads to numerous errors.

Be general. Don't make the program so specific that any change requires a major overhaul. Leave yourself some leeway to correct errors and implement better methods.

Arrange operations in a straightforward order. Programs are easy to follow and understand if they perform tasks in a sensible order without too much jumping around. Don't do a little of one task, interspersed with a little of another. Instead, try to do things in the order that a reasonable person would expect.

Use meaningful names. A name always means more than a number, but it is particularly helpful if it suggests the function or the purpose of the item it represents. So use P or PI for π, T or TEMP for temperature, N or NP for net pay, etc.. Here versions of BASIC that limit the user to one or two character names are inferior to more powerful versions of BASIC, FORTRAN, PASCAL, or assembly languages. The use of names makes changes easier, since all you must change are the definitions.

Group similar things together as much as possible. Keep all the inputs, outputs, definitions, etc., together. That way you know where to find them when you have to make changes or look for errors.

Keep different things separated. Don't use the same or similar names for entirely different things. Especially remember the obvious confusion between the numbers 0, 1, and 2 and the letters O, I, and Z. If you have names or other features that you find confusing, change them right away rather than wait until you make a mistake.

Avoid obscure or special features. Use the simplest features of whatever computer or language you are using. These features will be the easiest to understand and the most portable if you have to use another computer or if you buy a new system.

Don't be too clever. There is no advantage in seeking the trickiest way to do something or using results left over from another part of a program. Remember that tricky methods can cause unexpected errors if you change one part of a program.

DEBUGGING

Finding errors in a program (usually called debugging) is the most aggravating stage of programming. Errors can be very mysterious and resist detection even when you use the most sophisticated methods. Of course, no one ever knows how long it will take to locate a particular error or to remove all the errors from a program. This is why we have emphasized making programs as easy to debug as possible. This is also why so many vendors offer debugging aids.

The aims of most debugging aids are simple: to pin down the error to a small section of the program and to make its effects (and therefore its source) obvious. Most programming errors are remarkably simple once you uncover them. Often you will find it incredible that you could have done something so silly and not noticed it immediately. The debugging aids provide far more details about what the computer is doing than do normal runs and therefore often bring the error out in the open where you cannot miss it. The idea is the same as placing water in a tire to locate a puncture or red dye in a pipe to find a leak.

In Chapter 2, we briefly mentioned the following aids which are usually included in a debugger.

Breakpoints let you stop the program part way through.
Traces print some or all of the intermediate results while the program is executing.
Single-step modes execute the program one step at a time, thus allowing you to see exactly what the computer is doing.
Dumps show you the contents of part or all of the computer's memory.

Breakpoints help the programmer zero in on the location of an error. In a simple approach, one places a breakpoint halfway through the program to determine whether the error is in the first half or in the second half. One can then similarly narrow down the location to a quarter, an eighth, a sixteenth, etc., of the program. One can also put a breakpoint at the end of the first subroutine, check it out, and then continue in the same way with the second subroutine, the third, the fourth, etc. Breakpoints help you locate an error roughly; that is, place it in one section or one subroutine out of a long program. You usually need other debugging aids to pinpoint the error or to see its effects in detail.

The following hints will help you use breakpoints effectively.

- —Be sure to put the breakpoint in the right place. You can easily put a breakpoint where it is either never reached or produces no useful information.
- —Be careful that the breakpoint does not itself introduce errors by disturbing the program.
- —Find out how your computer resumes programs after a breakpoint. Methods for resuming programs vary greatly.
- —Remember to remove breakpoints that you no longer need.

Traces are used to display intermediate results in the hope that they will make the source of an error obvious. A simple trace lists the value of every variable

after each statement in the program. This allows the programmer to see where the error starts and thus to identify its source. The problem with a simple trace is that it usually produces a huge amount of information, most of which is either irrelevant or repetitive. The programmer may have to examine many pages of output to find a single incorrect number. Most traces let the programmer decide which variables are to be listed and how often their values are to be displayed or printed. This reduces the amount of information, but forces the programmer to determine what he or she needs.

The following hints will help when using traces.

- —Limit the trace as much as possible. Start by tracing only a few variables and printing their values infrequently.
- —Use breakpoints to limit the amount of tracing.
- —Determine what you're looking for before using the trace, otherwise you might end up with a lot of scrap paper.
- —Make the computer label all the information, otherwise you might have to go back to the listing to determine what the numbers mean.

A single-step mode helps the programmer pinpoint an error. In this mode, the program is executed one statement at a time, so the programmer can examine each step and see precisely what is happening. This is helpful only when you have used breakpoints or traces to pin the error down to a short section of the program. Otherwise the single-step mode is excruciatingly slow. At best, the single-step mode is tiresome and should be used only when faster methods have failed.

A dump shows many memory locations at one time. This lets the programmer see if sets of data are being loaded or processed correctly. Sometimes, by printing everything at one time, a dump can make the source of the error obvious; typical problems are data being loaded from the wrong place in memory or stored in the wrong place, values that are all the same or all zero, values that are all one position away from where you thought they were, and values all in error by a constant amount such as one or two. Dumps, like traces, often produce a great deal of useless information. The programmer can easily get a headache trying to extract what is important. A complete memory dump has traditionally been the programmer's final recourse; it typically provides time for a coffee break and a large supply of paper that can be used for making travel plans and preparing for a new career.

Newcomers to the computer field often wonder why debugging programs is so difficult and time-consuming. They ask, "Why don't you do it right the first time?" There are two major reasons why programs contain so many inexplicable errors. One is that computers are extremely literal; they do exactly what you tell them to do, no more and no less. They have absolutely no common sense and they do not improve with practice. A computer is like the popular comic character who follows orders to the letter with hilarious or disastrous consequences. Programmers find it difficult to be as precise, as detailed, and as complete as computers require them to be.

The other major reason for errors is that computers are complex systems. They usually involve many components and programs, much like a large bureaucracy. Writing a new program for a computer is like passing a new law or

establishing a new regulation for a government agency; the changes may reverberate through the system and produce results that are nothing like what you would expect. For example, your program may want to use part of the memory that the operating system needs or it may try to load some data that is not in the form that an I/O driver expects. The result is chaos; the computer is working against itself, much as one government agency may be building a new highway through an area that another agency is attempting to restore for its historical value or preserve as a park. Such conflicts are difficult to resolve because the tools one uses to identify or correct problems often contribute to them instead. The key to avoiding these conflicts is to have good systems software and documentation, just as the key to avoiding conflicts in business or government is to have good managers and well-defined procedures.

Where Is The Error?

Beginning programmers often throw up their hands when their programs produce incorrect results. Their usual questions are What do I do now? Didn't the computer say the program was right? Is something wrong with the computer?

The answers to these questions are

•Sit down and plan how to find the mistake.

•A translator (such as a BASIC interpreter) tells you only if the program follows the rules of the language. It does not tell you whether the program is sensible or will make the computer do what you intended. After all, the instruction "Remember to let the dog in and close the refrigerator" is grammatically correct and may even sound right, but could result in a very cold dog if the person who carries it out is like a computer and has no common sense.

• The computer does make a few mistakes on its own, but these are very few indeed when compared to the number made by programmers.

• Experienced programmers know where errors are likely to occur. Often a simple checklist can help you locate a puzzling mistake. Start by asking the following questions.

Did you include everything in the program? Does every section of the design appear in the listing? It's surprisingly easy to omit an entire section.

Are all variables either given initial values or read from an input device? You cannot assume that a variable is zero just because you have never given it a value. The computer's memory may contain all kinds of junk, particularly after you have run several programs.

Is there enough input data? Computers can do strange things when there isn't enough data to satisfy their appetites.

Do all branches have the correct conditions and the correct destinations? It's very easy to get things backwards (e.g., branch if A = B instead of A ≠ B) or branch to the wrong place.

Does the program work properly in trivial cases? Situations where there is nothing to be done or no data handled often are never considered, even by experienced programmers. What happens, for example, if a payroll program starts

to process temporary employees and finds that there weren't any during the current pay period?

Do the first and last iterations of each loop work properly? These iterations are usually special and therefore worth checking by hand, whereas all intermediate iterations are the same.

Is the data being read in the correct order? Remember that the computer does things in the order you specify. If you have the computer print the input data immediately, you will know that it has at least been read correctly.

Are the results printed or displayed in the correct order (or at all)? The computer will not print anything that the programmer does not specifically request and it will produce results in the specified order.

Are the results changed before they are printed? The computer will only print the latest values. One danger is to use the same name twice for different purposes. This is a problem in BASIC where the choice of names is limited.

Have you confused things that look similar? The variable X and the variable X1 are entirely different as far as the computer is concerned; so are memory address 1000 and the number 1000. Check which one you meant to use.

Does the program work correctly when the result is zero? Often we know what the computer should do when the result is positive or negative, but not when it is zero. For example, a billing program should send the customer a bill if his or her balance is positive and a refund if the balance is negative. But what should it do if the balance is zero?

Are things going in the right direction? It is easy to type A = Y when you mean Y = A or to calculate R − 1 when you mean 1 − R. Incorrect ordering is more difficult to recognize than an omission because a quick glance shows whether all the terms are present.

Have you forgotten to tell the computer to skip statements that it should not execute? Remember, the computer plows straight ahead unless you tell it to detour around things.

Have you given the computer enough memory space to use? You may have to specify the size of the sets (e.g., in BASIC, with DIM statements) or the maximum size of a program or data area. Be sure that you allow more space than you're ever likely to need.

Are the numbers too large or too small? A computer can only handle numbers in a particular range (see your manual). It may confuse tiny numbers with zero; this can result in a disaster if it later tries to use the numbers as divisors.

No matter what debugging aids you have, a simple checklist can often be the best way to find errors. Check the obvious things; don't try to go through long or complex calculations by hand—you're likely to make more errors checking them than you did programming them. A checklist, however, can get you to the point where the debugging aids will really be valuable.

A Debugging Example—Finding the Largest Number

The following example describes the checking and debugging of a BASIC program. The program finds the largest element in a set of data values entered from a keyboard. The initial version, derived from the flowchart in Figure 4-11, is

```
 10 REM ASK OPERATOR FOR NUMBER OF ELEMENTS (N)
 20 PRINT "HOW MANY ELEMENTS"
 30 INPUT N
 40 FOR I = 1 TO N
 50 REM ASK OPERATOR TO ENTER THE NEXT ELEMENT (E)
 60 PRINT "WHAT IS THE NEXT ELEMENT"
 70 INPUT E
 80 REM IS THIS ELEMENT THE LARGEST ONE SO FAR?
 90 IF E > L THEN 120
100 REM YES, SAVE IT (AS L) FOR LATER COMPARISONS
110 LET L = E
120 NEXT I
130 PRINT "LARGEST ELEMENT =", N
999 END
```

We ran this program for the simple case of a single element with a value of 50; that is, we entered the value 1 for N (the number of elements) and the value 50 for E (the first and only element). The result was

```
LARGEST ELEMENT = 1
```

Where in the world was the computer getting a silly answer like 1? It executed the program, so we didn't violate any of the rules of BASIC. Maybe more information will help. Let's introduce a tracing statement that will print all the values; such a statement is

```
75 PRINT "I =", I, "E =", E "L =", L, "N =", N
```

Running the program with this insertion (and entering the values 1 for N and 50 for E from the keyboard) produced the following intermediate results:

```
I = 1  E = 50  L = .5  N = 1
```

The values for the variables are, therefore, going left to right: I = 1(first iteration), E = 50 (the value we entered from the keyboard), L = .5 (oops!), and N = 1 (the other value we entered from the keyboard). So something is wrong with L. Checking the statements before line 75 by hand shows that none of them even mentions L. Thus the value printed by line 75 is whatever the computer finds when it looks up L. We can correct this by inserting a new line (a new 75) that assigns the value E (the current element) to L the first time through (when I equals 1). The new line is:

```
75 IF I = 1 THEN 110
```

But why is the computer telling us that the maximum value is 1 (not .5, which is the supposed value of L). Examining the PRINT statement on line 130 tells us the reason: we accidentally typed N instead of L so the computer printed the value of N. Note that the computer does exactly what we tell it to do; it does not check to see if that makes sense or corresponds with the printed explanation. The programmer must make the computer's automatic operation produce the results he or she wants.

The corrected program is

```
 10 REM ASK OPERATOR FOR NUMBER OF ELEMENTS (N)
 20 PRINT "HOW MANY ELEMENTS"
```

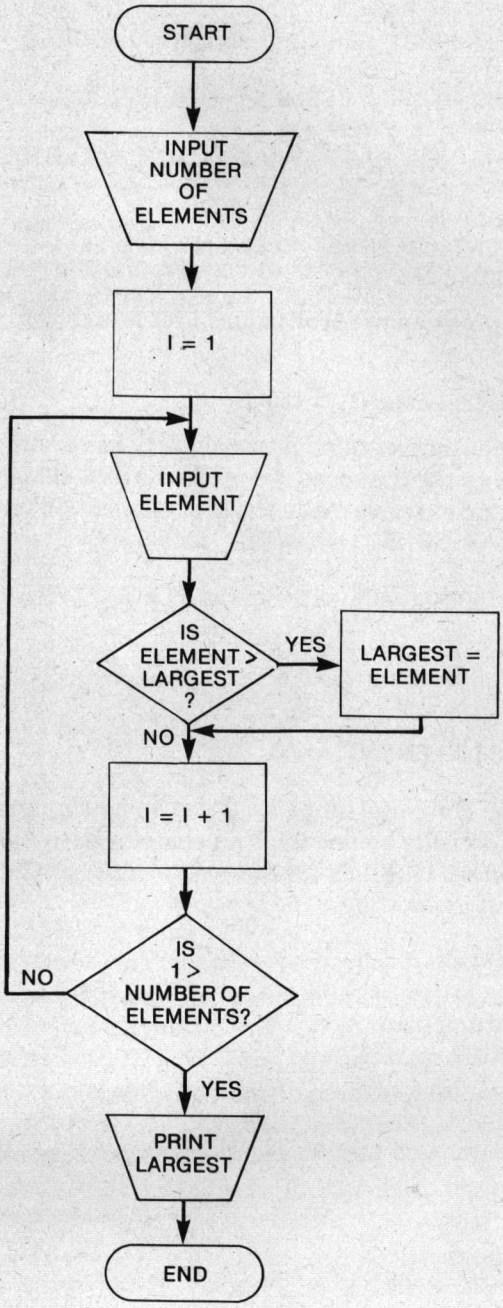


Figure 4-11. Rough flowchart for finding the largest element

```
30 INPUT N
40 FOR I = 1 TO N
50 REM ASK OPERATOR TO ENTER THE NEXT ELEMENT (E)
60 PRINT "WHAT IS THE NEXT ELEMENT"
70 INPUT E
```

```
 72 REM FIRST ELEMENT IS INITIAL GUESS AT LARGEST
 75 IF I = 1 THEN 110
 80 REM IS THIS ELEMENT THE LARGEST ONE SO FAR?
 90 IF E > L THEN 120
100 REM YES, SAVE IT (AS L) FOR LATER COMPARISONS
110 LET L = E
120 NEXT I
130 PRINT "LARGEST ELEMENT =", L
999 END
```

Running this program with the values 1 for N and 50 for E produced the obvious result

```
LARGEST ELEMENT = 50
```

Now we tried the program on two slightly more complicated cases, each with two elements. In the first case, we made the first element 50 and the second one 60; in the second case, we made the first element 60 and the second one 50. The results were

Case 1

```
LARGEST ELEMENT = 50


LARGEST ELEMENT = 50
```

Somehow, the computer is picking the smaller number instead of the larger number. The decision on line 90 is an obvious place to look for an error like this. Let's add another tracing statement so we can see the intermediate values; the new statement is

```
85 PRINT "I =", I, "E =", E, "L =", L, "N =", N
```

The results of the trace were

Case 1

```
I = 2  E = 60  L = 50  N = 2
```

Case 2

```
I = 2  E = 50  L = 60  N = 2
```

Everything seems alright, so let's check the program by hand with the values from the trace.

Case 1

E = 60 and L = 50, so E is greater than L and the computer proceeds to line 120 and exits from the FOR–NEXT loop.

Case 2

E = 50 and L = 60, so E is not greater than L and the computer proceeds to line 110, sets L equal to E (50), and then exits from the FOR-NEXT loop.

Obviously, we have things backwards. We want the condition in the IF-THEN statement (line 90) to be "IF L > E", not "IF E > L".

Our final program is therefore

```
 10 REM ASK OPERATOR FOR NUMBER OF ELEMENTS (N)
 20 PRINT "HOW MANY ELEMENTS"
 30 INPUT N
 40 FOR I = 1 TO N
 50 REM ASK OPERATOR TO ENTER THE NEXT ELEMENT (E)
 60 PRINT "WHAT IS THE NEXT ELEMENT"
 70 INPUT E
 72 REM FIRST ELEMENT IS INITIAL GUESS AT LARGEST
 75 IF I = 1 THEN 110
 80 REM IS THIS ELEMENT THE LARGEST ONE SO FAR?
 90 IF L > E THEN 120
100 REM YES, SAVE E (AS L) FOR LATER COMPARISONS
110 LET L = E
120 NEXT I
130 PRINT "LARGEST ELEMENT =", L
999 END
```

This program handles our test cases correctly.

You should learn the following lessons from this debugging exercise:

- —The computer never told us explicitly that we had made mistakes. It simply produced the wrong results. The programmer must know what the results should be and must make sure that the program operates correctly. Correct operation means more than correct use of language.
- —Traces and other debugging aids help by providing detailed information about what the computer is doing. This information can make the source of the error obvious.
- —The most likely programming errors are things that beginners find confusing. Forgetting to initiate variables, getting the logic backwards, and confusing things that look alike are common problems. The computer always does what it is told, but programmers can easily tell it to do the wrong thing when the procedure is long, complex, or confusing. Effective debugging of programs requires an understanding of human behavior as well as computer behavior.

Keys to Debugging

If your program doesn't work, don't panic. Instead, proceed logically.

- —Be sure that everything is in the program.
- —Try the simplest possible case. Keep adding more complexity until the error occurs.

—Pin the error down by checking sections of the program.
—Add trace statements so you can see the intermediate results.
—If you still can't find the error, check the program by hand.

Some errors are truly mystifying. If you encounter one of those, try the following:

—Trace each step, starting from the beginning.
—Ask someone else to look at the program. A fresh viewpoint often finds the problem right away.
—Take a break. You're probably just making your head swim at the moment and will undoubtedly do better later. Debugging requires patience, common sense, and a clear head.

TESTING—DOES THE PROGRAM REALLY WORK?

Testing is closely related to debugging. The objective of testing is to determine whether the program satisfies the requirements you defined for it. It will certainly not satisfy those requirements if it still contains errors.

Testing, however, is more than just finding errors. Testing a program ensures that it performs up to certain standards, as flight tests and road tests ensure that airplanes and automobiles perform up to standards. As with other tests, the idea is to try the program under realistic conditions. The problem for the programmer is to determine what those realistic conditions are. In the case of the automobile, General Motors and Ralph Nader surely have different conceptions of a thorough, realistic test.

The ideal test would involve trying every possible case. But this ideal is usually impractical, since there could easily be billions of cases. The programmer must try to provide the highest level of confidence in the program that can be achieved with a reasonable expenditure of time and effort.

Guidelines for Testing

We suggest the following approach to testing.

Start thinking about testing early. Consider both test cases and methods for making it easy to perform tests and observe the results.

Check all trivial cases. These include zero inputs, no data, empty files, and no tasks to perform.

Check all special cases that the program singles out for whatever reason.

Try at least one case for each path through a program.

Choose test data randomly if you have no basis for selecting it. Random number tables are commonly available in books and on most computers. People often have an unconscious bias in favor of small numbers, even numbers, or lucky numbers.

Choose the simplest possible cases for each test. Don't include any unnecessary complications.

Try all extreme cases, such as the largest and smallest values of variables.

Watch for situations where a result could be zero. If a program is comparing numbers, be sure that it works correctly when the numbers are equal.

Be careful of situations where a person may enter improper data values, respond incorrectly to questions, or follow the wrong procedures. Always expect the worst. You may want to let an inexperienced person try the program.

Watch for situations where the program depends on the order of data entries or assumes that the entries are positive, nonzero, or unequal.

Record the tests you ran. Then you will know what has already been checked if errors are discovered later.

Plan your testing carefully. Try to include cases that you know from experience are likely to occur.

Remember, you are not trying to find every last error. The aim of testing is to provide a full measure of quality, not to guarantee perfection. For example, in testing the program that finds the largest element in a set (Figure 4-11), we would want to try such cases as

- —No elements in the set
- —Only one element
- —Two elements with the larger value first
- —Two elements with the smaller value first
- —A random number of elements (between three and the largest number allowed) with the values obtained randomly

A random case avoids the problem of unconscious bias. This series of tests does not ensure that the program always works, but imagine how many trials exhaustive testing would require.

DOCUMENTATION

Documentation is an often neglected stage of programming. Designing and coding programs can be creative and enjoyable; debugging and testing them can be frustrating but challenging; documentation, on the other hand, is tedious, detailed work. Good documentation, however, is an essential part of a useful program. Even the person who wrote the program will soon forget precisely how it operates. Others can only guess at what the programmer was trying to accomplish. Those who want to use a program, correct errors in it, or make it do new jobs need good documentation to have a fighting chance.

People may want to know many things about a program, such as:

Purpose
Inputs required
Outputs produced
Basic outline of how it works
Order in which it performs operations
The way it handles data
Parameters that can be varied (and how they can be varied)
Amount of memory needed and its location
Peripherals required
Other programs that may be needed
Error messages and error handling

Execution time
Language used
Limits on amount of data or values of paramenters
Accuracy
Special cases

Many tools can help with documentation. Flowcharts can show how the program works and how it handles data. Parameter and definition lists can describe the variables used and how they can be changed. Memory maps can describe the memory requirements of the program. Comments can explain what parts of the program do. Library forms can provide a standard way to describe subroutines and entire programs.

Flowcharts serve as a pictorial form of documentation as well as design. As in design, a flowchart is best used to provide a rough description of the program. Flowcharts rapidly become cluttered and difficult to read if they contain too many details. As we mentioned previously in discussing design, several types of flowcharts may be useful, for example, general flowcharts that show how the overall program works, specific flowcharts that describe parts of the program in greater detail, and data flowcharts that show how the program deals with particular types of data.

Parameter and definition lists are a useful way to collect information about the program. These lists are often divided into categories such as input variables, constants, definitions, input and output devices, and subroutines. The program user then knows where to look for specific information. These lists should appear at the beginning of the program; descriptions of parameters and definitions that are scattered throughout the program create endless problems for the programmer as well as for the user.

Memory maps show what memory locations the program uses. Here again a picture (see Figure 4-12) is convenient and informative, particularly if the addresses are discontinuous. The program documentation should also tell whether the program can be moved (or relocated) and how.

Comments are statements the translator (assembler, interpreter, or compiler) ignores. All they do is show up in the program listing and thus allow the programmer to combine documentation with the statements that are actually translated. Comments are like captions for pictures or footnotes in technical articles. In theory they should explain what the program is doing in terms that anyone could understand; in fact, few programmers use comments effectively. We suggest the following rules:

—Comments should explain the purpose of sections or statements, not simply repeat formulas or describe features of the computer language.
—Comments should be readable. Nonstandard abbreviations should be avoided. The extra typing will be more than offset by a large increase in clarity.
—Comments should be limited to important points. There is no need to explain the obvious.
—Comments should use a standard form and standard terminology. Don't use two names for the same thing, even if the names are similar. Repetition is no vice here.

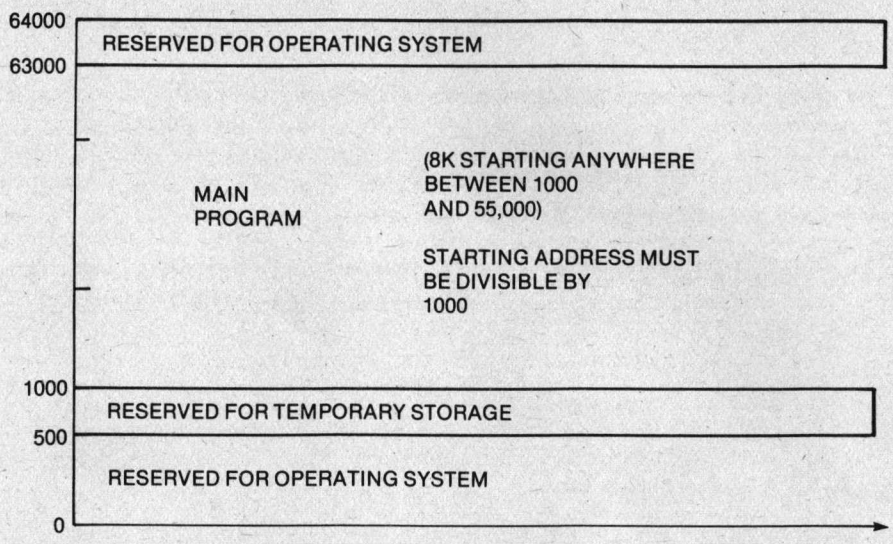


Figure 4-12. A typical memory map. This program can be relocated within a certain range of addresses but it also requires some fixed addresses.

- —Comments should be brief and to the point. A complete description should be left to other forms of documentation.
- —Comments should be clear. Ask yourself whether or not you would understand the comment if you were only casually familiar with the program.
- —Comments should be kept up to date. When you change the program, change the comments also. Outdated comments are worse than no comments at all.
- —Place the comments immediately ahead of the section of the program they describe. Move the comments when you rearrange the program.
- —Separate the comments from the program. Blank lines around the comments make the program far less crowded and easier to read.
- —Don't have too many comments. Sometimes you can lose track of the program itself. The comments should describe the program, not swallow it up.

Library forms provide a standard way to describe subroutines. Figure 4-13 shows a typical form. These forms can be conveniently collected in a loose-leaf notebook and saved for future reference. You may want to organize the forms by subject, such as mathematical routines, string handling, statistical analysis, I/O routines, and utilities.

Guidelines for Documentation

We suggest the following approach to documentation.

Work on documentation as you design, code, debug, and test the program. Don't leave it all to the end. In fact, good documentation will help you debug and test programs.

Title:

Purpose:

Language

Initial conditions:

Final conditions:

Requirements:

Memory:
Time:

Sample case:

Listing:

Figure 4-13. A typical library form

Improve the documentation as you work on the program. It certainly will never improve by oneself.

Look at the documentation from an outsider's point of view. Would you really be able to understand and use the program if you hadn't written it? It is, of course, far easier to complain about other people's documentation than actually to produce good documentation oneself.

Have someone else look at the documentation. The writer of the program is often too close to it to determine what documentation the user will need.

Establish a standard documentation package. Extra time spent here will eventually pay off even if it seems to be wasted.

Complete the documentation before you go on to the next task. It's easy to postpone an uninteresting job and never return to it. Furthermore, the program will not be nearly as clear and familiar later.

Produce different levels of documentation for users with different needs. A general flowchart and comments at the head of each section will be sufficient for casual users. Those who must change or extend the program need more details, but casual users should not have to dig through them.

SUMMARY

This chapter has dealt with all the stages of program writing. The precautions that apply to most human activities apply here as well. The programmer who plans carefully, works cautiously and systematically, and documents each step will be far ahead in the end. In the design stage, such methods as flowcharting, modular programming, structured programming, and top-down design are valuable tools. In other stages, though, there are few formal methods and common sense must be your guide. We have tried to provide some helpful hints and guidelines. Writing programs can be challenging, enjoyable, and productive if you proceed with caution and try to get the most return from your efforts.

REFERENCES

Kernighan, B.W., and P.J. Plauger. *The Elements of Programming Style, 2d ed.* New York: McGraw-Hill, 1978.

Leventhal, L.A. "The Top-Down Approach," *Kilobaud,* May 1978, pp. 68–73.

Leventhal, L.A. "Why Structured Programming?" *Kilobaud,* February 1978, pp. 84–88.

Nagin, P., and Ledgard, H. F. *BASIC with Style: Programming Proverbs.* Rochelle Park, N.J.: Hayden, 1978.

Van Tassel, D. *Program Style, Design, Efficiency, Debugging, and Testing, 2d ed.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

5 Peripherals

A computer can do nothing by itself. A useful computer (like the one in Figure 5-1) must have peripherals that can load data and programs, display results, and save information for later use.

In this chapter, we describe peripherals in detail, discussing input devices, output devices, and storage devices. We emphasize standard peripherals—keyboards, video displays, cassette recorders, disks, and printers. We mention only briefly less common peripherals such as voice I/O devices, paper tape readers, analog I/O systems, music generators, and plotters.

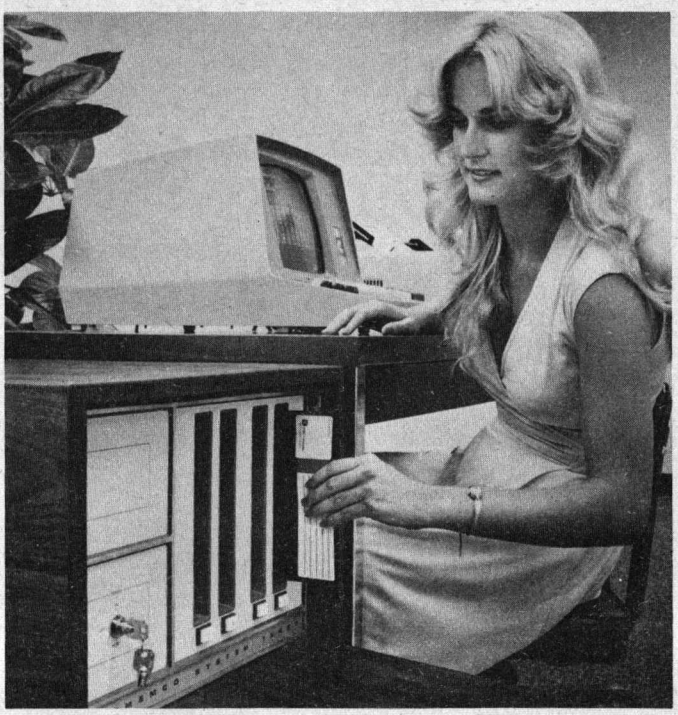


Figure 5-1. The Cromemco System Three computer, a Z80-based system with two to four 8-inch floppy disks (Courtesy of Cromemco, Inc., Mountain View, CA)

We will answer the following questions.

—What peripherals must a computer have?
—What features do typical peripherals have?
—What options are available and how are they used?
—What are the tradeoffs among cost, performance, and ease of use?
—How can you select peripherals to meet your needs?

We leave interfacing and maintenance to Chapters 6 and 7, respectively.

STANDARD PERIPHERALS

A computer must have an input device, an output device, and a mass storage device. The standard peripherals for personal computers are keyboards for input, video displays and printers for output, and cassette recorders and floppy disks for mass storage. These devices differ from the ones most people associate with computers. Keyboards have replaced card readers and keypunches; video terminals have replaced flashing lights; the cassette recorder or floppy disk has replaced the magnetic tape units that appeared in so many movies and television programs. Cassettes and diskettes have replaced the famous punched cards that introduced most people to computers. Today's personal computers are ordinary in appearance (see Figures 5-1 through 5-4 for typical examples); they are small and few of their peripherals are exotic.

THE KEYBOARD — A SIMPLE INPUT DEVICE

Keyboards are the primary input devices (and control panels) for most personal computers. In fact, many computers (Figure 5-2) have only one or two switches (usually ON/OFF and RESET) and even those may be concealed. The keyboards vary from small ones with ten or twenty keys to large ones that resemble a standard typewriter. Most people have seen small keyboards in electronic cash registers, credit verification terminals, and automatic banking terminals, and large keyboards at airline ticket counters, banks, car rental agencies, and government offices.

There are unfortunately no standards for computer keyboards although most are like typewriters. Among the variations are size, number and placement of keys, availability and meaning of shifts (many keyboards have only capital letters and use shifts to provide additional characters), coding, and availability of control keys and indicators. Inexpensive keyboards usually have fewer keys and are not as easy to use as more expensive keyboards.

One important characteristic of a keyboard is what code it produces. The computer does not, of course, read text directly or look at the keytops. It cannot recognize the patterns that form a handwritten or typed letter, digit, or punctuation mark. As we mentioned in Chapter 1, the computer only handles numerical values that are assigned to the characters, much as sequences of dots and dashes are assigned to letters and numbers in Morse code. The way in which these numerical values are assigned is called a *code*; the most common code is the American Standard Code for Information Interchange or ASCII (see Appendix 2).

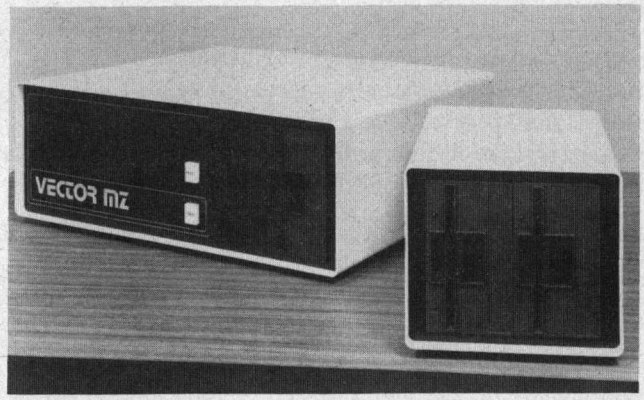


Figure 5-2. The Vector Graphic MZ computer and MICRO-STOR add-on floppy disk system. The MICRO-STOR can be used with any S-100 bus computer. (Courtesy of Vector Graphic, Inc., Westlake Village, CA)

Figure 5-3. The Exidy Sorcerer home computer (Courtesy of Exidy, Inc., Data Products Division, Sunnyvale, CA)

Most packaged computers include an ASCII keyboard and the user need not be concerned with how it works. But you must be careful of advertised keyboards that use some other code or none at all. Such keyboards may resemble the ones supplied with computers and may be available at bargain prices, but they are useless unless you can add the electronics that will produce ASCII.

Many features can make a keyboard easier to use, such as:

Keys that are organized and spaced like a standard typewriter (see Figure 5-3). Clearly this organization lets an experienced typist work faster.

A numeric pad. This is a separate set of ten keys for the decimal digits, usually organized like the keys of an adding machine. The computers in Figures 5-3 and 5-4 have numeric pads to the right of their regular keyboards. Numeric pads let

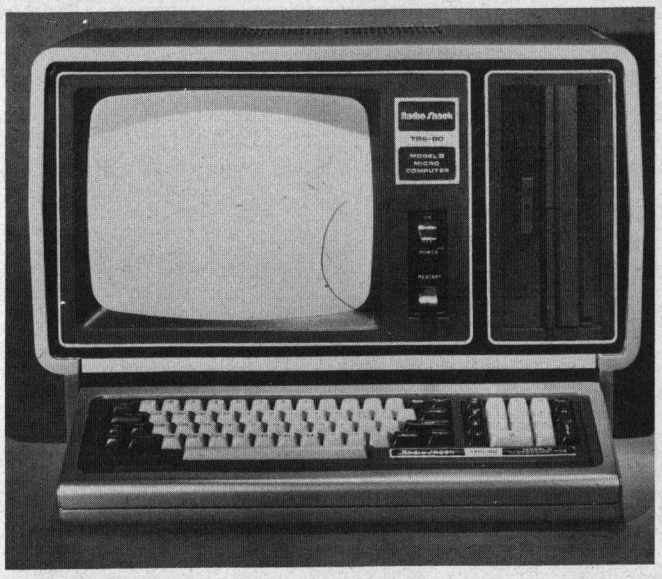


Figure 5-4. The Radio Shack TRS-80 Model II Microcomputer (Courtesy of Radio Shack Division of Tandy Corp., Fort Worth, TX)

you enter columns of figures much faster, since the digit keys are difficult to reach on a standard typewriter.

Well-placed control keys. Control keys perform housekeeping functions. They may include shift keys, line feed (to skip a line on the page), carriage return, backspace, delete, and tab keys. Their placement is important, particularly because no standards exist. Common control keys should be convenient but not so close together that users often reach for one (such as ENTER or RETURN) and press another (such as CLEAR or DELETE) instead. In Figure 5-4, the black keys on the outside of the keyboard and the numeric pad are the control keys.

Lighted indicators. These show the keyboard's mode of operation and whether it is actually connected to the computer. A keyboard that is directly connected to a computer is said to be *on-line* while one that is being used to prepare data or text for later entry into the computer is said to be *off-line.*

Rollover. This allows the user to press several keys without waiting for the earlier ones to bounce back. Rollover greatly increases the rate at which a skilled typist can work.

Good tactile feel. Like typewriters, some keyboards simply feel better than others. The touch on most computer keyboards is rather soft. If you learned to type on a manual typewriter, you should lighten your touch to make the computer keyboard last longer.

Some applications require a keyboard with certain features. Word processing, for example, requires a keyboard with both upper and lower case letters; however, a numeric pad is not essential. Order entry with a video terminal requires special function keys for common operations like clearing the screen, moving from one part of the order form to another, and transferring a completed entry to the computer or storage device. A numeric pad is essential for entering

columns of figures, as in maintaining inventory records. Line and figure drawing (graphics) requires a special keyboard; for example, the keyboard of the Exidy Sorcerer (Figure 5-3) has keys that produce segments of straight lines, circles, and other geometric forms.

Keyboard construction is important in any application. A good keyboard has a solid foundation, high quality keys, and is easy to clean and service.

Unlike typewriters, computer keyboards are not directly connected to a printer. When you press a key on a typewriter, you immediately see the printed result; when you press a key on a computer keyboard, all that happens is that the computer receives the corresponding code. Everything else requires extra hardware and software, even if the computer has a printer or video display. The data must be received from the keyboard and sent to the output device (or *echoed*) before anything is printed. Unless this happens, the user sees nothing. Most computers provide visual feedback, but some do not, and all require practice and a break-in period. At first, you may find yourself waiting for the familiar sound of the typewriter; the experience is like talking without being able to hear your own voice.

Advantages and Disadvantages of Keyboards

Most of the advantages of keyboards are obvious. They are easy to use and allow rapid data entry if you can type. Changes and corrections are simple, particularly if your keyboard is connected to a video display and you get immediate feedback. Certainly keyboard entries are easier to correct than are entries on cards or paper tape. Keyboards are also quiet, inexpensive, electrically and mechanically simple, highly reliable, small, and portable. They provide direct input to a computer without using any media such as tapes or cards.

Most of the disadvantages of keyboards are equally obvious. Only one person can use a keyboard at a time and the user ties it up, particularly if he or she is a hunt-and-peck typist. The user must type well to enter data at a reasonable rate. Keyboard entry is thus a problem in schools where there are typically many students per computer and few who can type. Since keyboards do not produce permanent records, the entries must be repeated unless they can be saved using some other peripheral. Other problems are the lack of standards for keyboards (particularly for keys not on typewriters), alphabet-dependence, and wear. Keys may become stuck and the tactile feel will deteriorate, especially if someone spills soda pop, coffee, or maple syrup on the keyboard.

In purchasing a keyboard, you must ask the following questions: Will it work with your computer? Is the proper interface available or included in the computer? Does it feel right? Is it pleasant to use? Is it organized in a way that is easy to use and familiar to you? Is it well-constructed? Can it be cleaned and serviced easily? Does it have the features required for your application, such as a numeric pad, rollover, upper- and lowercase letters, graphics symbols, control keys, and function keys? Can you add these features if you purchase the keyboard without them?

Some one-board computers have small, calculator-like keyboards. These are inexpensive and durable, but they limit the way in which data can be entered since a full typewriter character set is not available. They are, however, a great improvement over old-fashioned panel switches and are satisfactory for short programs when better input devices are not available.

THE VIDEO DISPLAY — A SIMPLE OUTPUT DEVICE

The standard output device for personal computers is the video display (see Figures 5-1 and 5-4). Some appliance computers use an ordinary television set to reduce cost. A special interface (known as a video converter or video modulator) is required, but it costs only $10 to $20. Unfortunately television sets do not make good video displays because the characters are usually difficult to read and only a few relatively short lines can be produced at a time. Other computers use a specialized version of a television set called a video monitor; these are like the monitors in store security systems.

The video or CRT terminal (see Figures 5-1 and 5-6) is a combination of a keyboard and a video display. Most terminals have more features and are easier to interface than video monitors or television sets, but are considerably more expensive. Like keyboards, video displays vary greatly in quality, features, and ease of use and interfacing, with a correspondingly wide variation in cost.

The characteristics of video displays include the following:

Number of lines on the screen. This determines how many lines of text you can see at a time. It typically ranges from 12 to 40, with the more expensive devices having more lines. A larger number of lines lets you handle longer forms and reduces the likelihood of losing information at the top or the bottom of the screen.

Size of screen. The most common sizes are 9-inch and 12-inch. Reading text from a small screen is as tiring as watching programs on a small television set. A large screen, however, may make the display bulky and awkward, the characters too large, and the spacing too wide.

Length of lines. This is specified (like a line of typing) in terms of number of characters. Typical values range from 32 to 80. Of course longer lines let you handle larger forms and see more data at one time. Television sets are limited to 40 or fewer characters per line.

Size and form of characters. Larger and better-formed characters are more legible.

Whether control characters produce any display. Some terminals indicate that control characters have been received.

Which characters are available. Some displays only have capital letters, while others have far more extensive character sets and may even provide special characters used in technical material or foreign languages.

What code is used. Most displays use ASCII.

Whether graphics symbols are available. Most CRTs can only produce typed characters. A special display or extra circuitry is needed to produce dots, squares, lines, and other geometric forms or notations.

Whether a cursor or pointer is available (for indicating the current active position on the screen) and how it can be controlled. Some terminals allow you to determine where the cursor is (it is said to be *addressable*) and allow you to move it anywhere on the screen. Other displays only permit you to move the cursor one space at a time or return it to its starting position (or *home*).

Whether scrolling is available. Scrolling (moving the text up or down on the screen) is convenient when you need to backtrack or when there is simply more data than the screen can hold at one time.

Many video displays or terminals also have special features that make them easier to use, such as:

A blink feature that can flash characters on and off for emphasis. This is often used to indicate errors, the position of the cursor, or the data currently being entered.

A hide (blank) feature that results in data such as passwords or sensitive information not being shown on the screen.

Inverse video that produces white characters on a black background for emphasis or improved visibility. Reverse video is similar, except that it inverts the normal way in which characters are displayed on a background, so it will produce black characters on a white background, if white on black is normal.

Highlighting whereby certain sections of the screen (such as the current line) are made brighter than other sections.

Protected fields whereby certain parts of the screen (representing forms or titles) cannot be changed from the keyboard. The user can fill in the form, but cannot write over it.

A status line that shows error conditions, operating modes, the current position of the cursor, and other information.

Data validation whereby data entries are checked to see if they consist of letters (as required in a name) or digits (as required in a date or unit cost), or conform to other specifications.

Advantages and Disadvantages of Video Displays

Video displays are popular because they are familiar, widely available, quiet, reliable, and easy to maintain. Since they have no moving mechanical parts, video displays can operate at high speed. There is no need to buy paper, ribbons, ink cartridges, paper holders, or other supplies and accessories. Besides, there is no need to load paper or ribbons, align paper, or deal with jams and other annoying mechanical problems. The lack of paper not only saves money, but also results in less confusion, better security, and a cleaner operation. Video displays thus even reduce waste and save trees and energy. After working with a keyboard and a video display, most users find printing terminals to be slow, noisy, difficult to use, and unreliable, like reading the script of a play or a television program without being able to see the action.

Video displays do have disadvantages. The limited size of the screen often means that you cannot see all the output at one time. Program listings and results may race by before you have a chance to examine them. Obviously this problem is more serious if your display can hold only a relatively small number of characters. Some displays have a special feature called *paging* that allows the user to examine one screen's worth of data (or *page*) at a time; others have facilities for slowing or stopping the display. No matter how many special features your display has, occasionally the lack of a permanent record (*hard copy*) is annoying. There is no easy way to save the output, study it later, or send a copy to someone. Video displays are also bulky; they take up a large amount of room and are awkward to move. Furthermore, they can produce glare and eye fatigue, particularly if you are not used to working on them or if the room lighting is inadequate.

Choosing a Video Display

In buying a video display, the user must ask the following questions:

Will it work with your computer? It must use the proper code and the proper interface and software must be available. Does it have a sufficient number of lines of sufficient length? What is the quality of the display? Are the characters distinct and clearly formed and are the lighting levels and contrast adequate? Is it easy to maintain? Can additional features be added, such as graphics or a printer interface?

Does it have all the features that your system software supports? The software cannot produce features that the video display does not have. For example, no computer can produce color graphics on a black-and-white television set, or control a cursor that does not exist, or produce graphics symbols that are not available. On the other hand, it is of no value for the display to have features that the software does not support. You will not get color graphics, for example, if your software makes no provisions for color. The video display must be compatible with the computer—too few features means that some of the computer's capabilities are wasted, while too many features means extra expense and complications for no benefit.

One source of confusion for the beginner is the use of the terms "video monitor," "video display," or "video terminal" to refer to circuit boards like the Vector Graphic Flashwriter II in Figure 5-5. These boards provide an interface between a computer and a video monitor or television set, but the user must provide the actual output device. In comparing prices and features, you must determine what is included, since the interface and the output device are often shown together but priced separately.

We have already noted that video displays vary greatly in quality and capabilities. Prices range from a few hundred dollars to several thousand dollars. A low price tag usually means a device with limited capabilities that may require extensive interfacing. Such a device may be ideal for users with

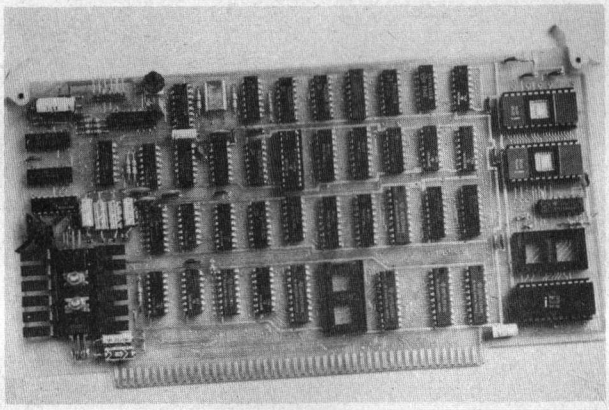


Figure 5-5. The Vector Graphic Flashwriter II video board provides 8 × 10 dot matrix characters for an 80 character by 24 line display with reverse video (Courtesy of Vector Graphic, Inc., Westlake Village, CA)

Figure 5-6. The Honeywell VIP7800 CRT terminal with numerous editing and other special features (Courtesy of United States Information Systems Group, Honeywell, Inc., Waltham, MA)

small budgets and adequate time, skill, and equipment. Those who intend to use the video display extensively will find a high-quality terminal with many features to be an excellent investment. Such a terminal will be much easier and faster to use and the extra expense will be more than offset by the added convenience and productivity.

Features like blink, inverse video, highlighting, and protected fields (called *attributes*) are particularly helpful in business applications. Blink, inverse video, and highlighting can call attention to important information, warn of errors, or single out the current line or entry. Protected fields make the screen look like a printed form. This makes it easy to enter invoices or orders and to process insurance forms, tax returns, attendance records, or reservations. Usually the computer will skip over the protected fields automatically, allowing the operator to enter simply one data item after another (such as invoice number, date, name of customer, address of customer, and payment terms). Clearly the operator can work much faster if the computer handles the alignment, spacing, and punctuation. Figure 5-6 shows an operator entering an order on the sophisticated Honeywell VIP 7800 terminal; note the use of inverse video, protected fields, and line drawing.

Color, despite its attractiveness, has so far found only limited use in practical applications. The main problems have been cost and lack of standards. Color, however, has obvious applications in art, graphics, and games, and can be used to highlight information in business, educational, and scientific applications.

MASS STORAGE DEVICES

Mass storage acts as a reserve for the computer's memory, allowing the computer access to additional programs and data. The medium on which you save

programs and data is often also the form in which you buy or trade them. As with records or tapes, the media can be stored in a library and only loaded into the storage device (or player) when needed.

The standard mass storage devices for personal computers are cassette recorders, floppy disks, and ordinary (hard) disks. Among the factors you must consider in evaluating mass storage devices are

- —How much storage does each unit provide?
- —How fast can information be stored and retrieved?
- —What access method is used?
- —How much does the device cost?
- —How complex are the interface and controller and how much do they cost?
- —How much do the media cost and how widely available are they?
- —Do recording and interface standards exist?
- —Are programs and data widely available in the required form?
- —Are the media reliable, permanent, reusable, and convenient to use? Can they be used on any computer?

CASSETTE RECORDERS

Cassette recorders are often used with personal computers (as shown in Figure 5-7) because the recorders are inexpensive, widely available, and easy to operate and maintain. The cassettes themselves are erasable, reusable, easy to store,

Figure 5-7. The introductory-level Radio Shack TRS-80 Microcomputer, with keyboard, 12-inch video display, cassette recorder, power supply, and User's Manual (Courtesy of Radio Shack Division of Tandy Corp., Fort Worth, TX)

and long-lasting (with a little care). Each cassette provides a large amount of storage in a familiar, convenient, and widely available form. Higher quality digital cassettes and recorders provide better performance than audio versions at a higher price.

The major disadvantages of cassettes are their low speed and serial access. The low speed means that a long program or large amount of data can take several minutes to load, tying up the computer while the user can do little except watch. Serial access means that there is no way to reach a place on a tape ahead of or in back of where you are without passing through the in-between places. Therefore, you often have to waste time playing or rewinding a tape just to reach a program or some data. Furthermore, the tape must be rewound after each use.

Cassettes have other disadvantages as well. There is no single standard for computer recording, so that cassettes recorded on one model of computer will not play on another model. In fact, cassette recorders vary enough in performance so that cassettes recorded on one computer may not even play on another computer of the same model; worse yet, your own computer may not be able to read your tapes unless you adjust the volume setting. Furthermore, cassettes have a high error rate and computers are less tolerant of errors than are entertainment systems. Cassettes also require manual intervention; the user normally has to control rewinding, playing, and recording as well as adjusting the volume setting for a particular tape.

The serial access problem with cassettes makes the use of short tapes desirable. While audio users prefer 60- or 90-minute cassettes for long recordings, computer users generally prefer 5- or 10-minute cassettes like the ones in Figure 5-8, each with a single program. Recording each program on a separate tape reduces the amount of searching but results in higher costs and a much larger number of cassettes to store and organize.

FLOPPY DISKS

In all except very slow and highly cost-sensitive applications, floppy disks are rapidly replacing cassette recorders as storage devices for personal computers.

Figure 5-8. The C-10 10-minute cassette for personal computer use (Courtesy of Verbatim Corp., Sunnyvale, CA)

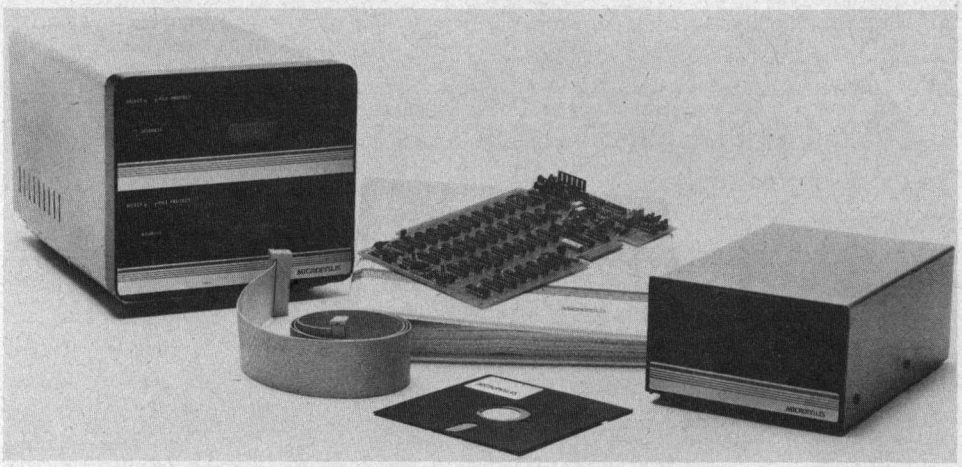


Figure 5-9. Single and double-drive models of the Micropolis MetaFloppy 5-inch disk family (Courtesy of Micropolis Corp., Northridge, CA)

The computers shown in Figures 5-1, 5-2, and 5-4 all use floppy disks for mass storage. The major advantages of disks over cassettes are their higher speeds and faster access times. Furthermore, the access is not serial since the head on a floppy disk system can be moved to any track on the disk, much as the needle on a record player can be positioned anywhere without playing the entire record. Floppy disk systems can be purchased with packaged computers (as in Figures 5-1, 5-2, and 5-4) or separately in single or double units (see Figures 5-2 and 5-9).

The increased speed of floppy disks means that programs and data can be loaded in seconds rather than in minutes, making loading a minor inconvenience rather than a major chore. Other advantages of floppy disks, besides random access (the ability to reach any position without passing through the in-between positions), are the limited number of standard recording techniques and the small amount of manual intervention required. The medium, called a *disk* or *diskette* (see Figure 5-10), is reasonable in cost (typically $3 to $10), erasable, reusable, easy to store and handle (special cases and files are now widely available), and long-lasting. Note how the operator in Figure 5-1 is simply sliding the diskette into its drive.

The disadvantages of floppy disks as compared to cassettes are their higher cost and greater complexity. Diskettes are more expensive and less widely available than audio cassettes. The systems and controllers (see Figure 5-11) are both more complex and more expensive. Furthermore, individual diskettes provide a relatively small amount of storage (as we will discuss later) and are susceptible to scratching, warping, and wear.

We should note that floppy disk systems vary greatly in their size, recording method, recording density, data capacity, type of controller, and transfer rate. We will not deal with all the variations, but we will mention the following.

—A mini-disk or minifloppy is 5¼ inches (13.3 cm) in diameter and can store 80,000 characters in normal or *single density.*

—A floppy disk, standard floppy, or full-sized floppy is 8" (20.3 cm) in

Figure 5-10. Various models of 8-inch floppy disks (Courtesy of Verbatim Corp., Sunnyvale, CA)

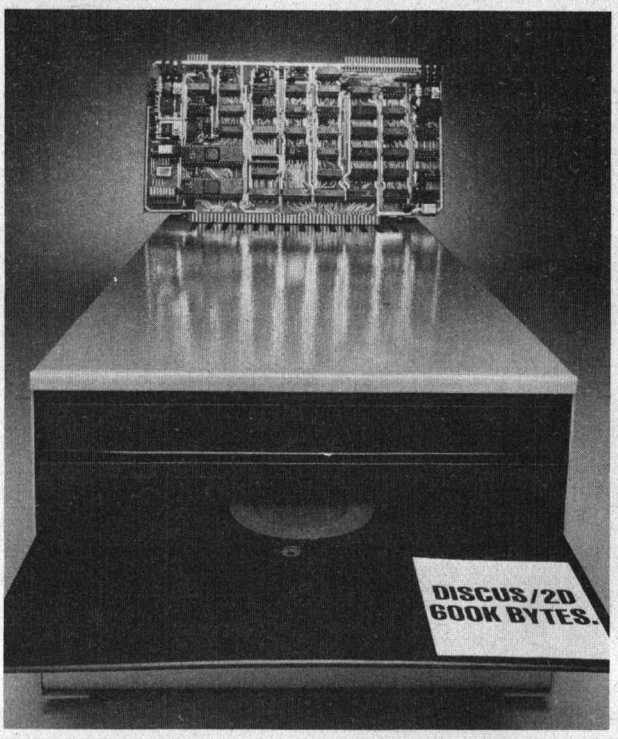


Figure 5-11. The Thinker Toys Discus/2D full-size floppy disk with controller (Courtesy of Thinker Toys, Berkeley, CA, a product of Morrow's Micro-Stuff)

diameter and can store 250,000 characters in normal or single density.
- —A double-density floppy disk can store twice as much data as a single-density disk (e.g., 500,000 characters on an 8" disk) but requires a special controller and software.
- —A double-sided disk can store data on both sides but requires special hardware, software, and media.

Despite the appearance of double-density and double-sided disks and other improvements in density, the basic problem with floppy disks is the lack of sufficient storage for many applications: 80,000 to 500,000 characters sounds like a lot, but you should realize that a typical typed page contains 2,000 characters. So 80,000 to 500,000 characters corresponds to 40 to 250 typed pages, assuming no space is wasted. Many short reports are that long, and few businesses would find that number of pages a significant fraction of their invoices or accounts; for example, even if an invoice was only ¼ of a page long, a single diskette could hold only between 160 and 1000 invoices. Even 1000 invoices is less than one month's activity for a small retail business or professional office. Most small businesses handle thousands of invoices per month and need immediate access to many months of records. The result is a large number of diskettes and a tremendous amount of time spent changing diskettes and copying them to ensure the retention of important records.

Clearly any disk (or cassette) containing data or programs must be copied as a precaution against wear, mishandling, accidental erasure or overwriting, fire or other natural disasters, and failure of medium or controller. Copies may also be needed for permanent records, use on other computers, or documentation. The more disks or cassettes there are, the higher the probability of discrepancies, mislabeling, and erroneous or unauthorized use.

Furthermore, the computer must have software that handles the transfer of data and programs to or from the disk. This software is called a *disk operating system.* However, the operating system itself requires disk and memory space, thus reducing the capacity of disk and computer to hold data and programs. The fact that the operating system usually occupies a large fraction of a diskette makes computers with one floppy disk awkward and difficult to use. To transfer data or programs to or from the disk, you need the operating system and either the data or programs (if you are going to load them) or some empty disk space (if you are going to save them). If you are going to copy a disk, you need the operating system, the original disk, and a blank disk. Obviously, at the very least you are going to spend a large amount of time changing disks. The result is that computers with one floppy disk are incomplete, like stereophonic systems with only one speaker. Packaged computers like those in Figures 5-1 and 5-2 therefore typically include two floppy disks. A third or fourth disk, as shown in Figure 5-1, provides extra storage and convenience, but is not nearly as important as the second disk.

You must be careful in buying a disk system. Minifloppies and single disk units are much cheaper than standard, double-disk units. Interfaces, controllers, and operating systems may not be included. If you have not used a disk before, you will have to learn what you really need and want.

HARD DISKS

Ordinary disks (also called hard, fixed, or fixed-head disks) have the largest capacity and are the most expensive of the mass storage devices commonly used with personal computers. Hard disks are the standard mass storage devices for larger computers; they are being used more widely with personal computers (often in cheaper, smaller forms) as the need for large amounts of storage becomes evident. The hard disks (see Figures 5-12 and 5-13) offer 5 to 70 million characters of storage, fast access (typically ten times the speed of floppies), and high reliability. Their disadvantages are high cost (usually at least two to ten times the cost of floppies) for controllers, mechanisms, and media; limited availability of media; and large, awkward size.

Fixed disks vary greatly in size, storage capacity, and features. Some, like the one in Figure 5-12, are 8 inches (20.3 cm) in diameter, the same size as a standard floppy disk (remember Figure 5-10). Others are larger, typically 14 inches (35.6 cm) in diameter. Sometimes the media itself can be removed and replaced and some controllers have the ability to handle several drives. In general, hard disks offer a large amount of random-access storage, enough to easily hold an operating system and thus make a single-disk unit practical.

Backup, however, is a problem with computers that have only one disk, particularly if the media cannot be removed. Users must keep their copies either on many floppy disks (creating an awkward copying and storage problem) or on

Figure 5-12. The Pertec Computer Corporation PCC 2000 small business computer with floppy disks and up to 40 megabytes of hard disk storage (Courtesy of Pertec Computer Corp., Los Angeles, CA)

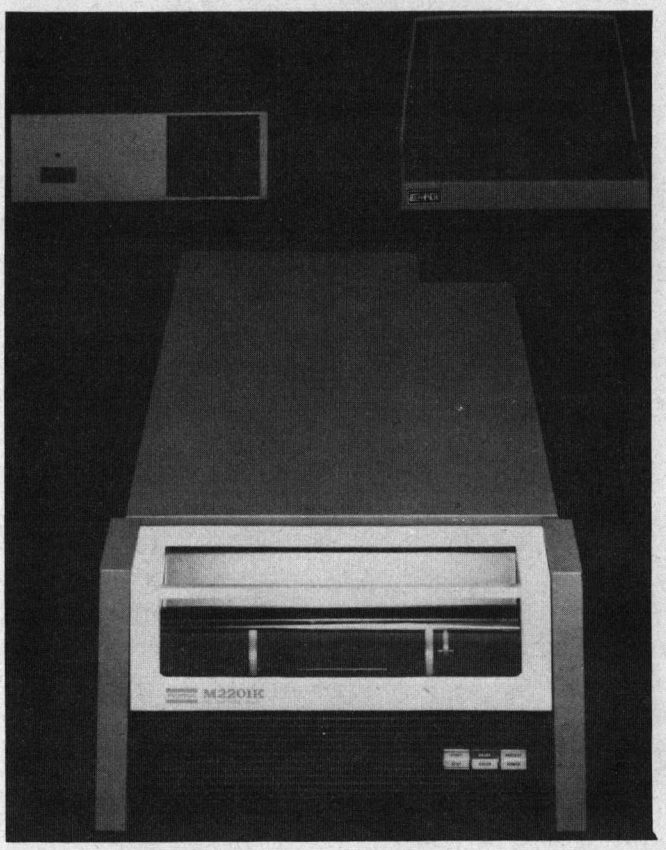


Figure 5-13. The Fujitsu M2201K disk cartridge drive. It offers 40 megabytes of storage per unit for S-100 bus computers. (Courtesy of MicroAge, Tempe, AZ)

magnetic tape (requiring an additional, expensive peripheral and rather slow copying). Presumably, manufacturers will find better solutions to the backup problem as the number of hard disks used with personal computers increases. For example, the computer in Figure 5-12 has two floppy disks besides its hard disk.

CHOOSING A DISK

There are many considerations involved in choosing any disk (fixed or floppy). Will it work with your computer? Is the proper interface available and at what cost? Is it compatible with your other hardware and software? Is an operating system included? What features does that system have, what software is available for it, and what peripherals does it support? How much storage does the disk provide and how fast can data be transferred to or from the disk? Does the system use standard media and a standard recording technique? Can additional disks be added? Can the storage capacity be increased and at what cost? What is the estimated lifetime of system and media? How much do the media cost and where are they available? How easy are they to store and handle? Can

the media be removed? Are there provisions for backup? Is recovery provided in the event of errors in the disk system? Good error recovery techniques minimize the loss of data and the repetition of transfers.

Disks have made computers more powerful and easier to use. Their availability with personal computers has made many business, scientific, and educational applications feasible, since they provide large amounts of easily and quickly accessed storage in a more convenient form than cards or tape. One problem that we have noted before is simple familiarity. Most people are accustomed to magnetic tape and punched cards, but not to disks. Once they overcome this unfamiliarity, those who work with disk-based computers soon find cassette and card-based computers to be as awkward and as inconvenient as manual typewriters, mechanical calculators, or old-fashioned telephone switchboards.

PRINTERS

The first peripheral most users add to a computer like the ones in Figures 5-1 and 5-12 is a printer. Obviously, you must have a printer for word processing and most business applications. You may, however, want one even if it is not strictly necessary. In applications like simple calculations, computer-aided instruction, and games, a printer provides a permanent record (hard copy) that you can study later, carry to a quiet room, save for reference or consultation, use as a final report or a completed assignment, or compare to earlier results. While some people prefer to work without hard copy, others are so accustomed to having it that its absence makes them uncomfortable.

We should emphasize, however, the importance of avoiding hard copy unless it is absolutely necessary. Paper is expensive and heavy use of it generally results in a large amount of waste, mess, and confusion. Furthermore, printers are slow, noisy, and relatively unreliable. We recommend that you do not buy a printer initially unless you absolutely need it and that you do not use your printer unnecessarily. Using the video display as your primary output device saves time and money, makes operations cleaner and more reliable, and reduces

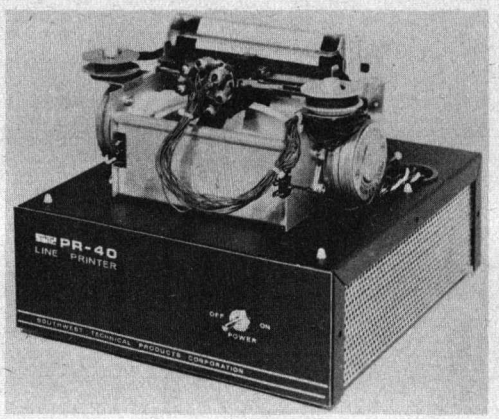


Figure 5-14. The SWTP PR-40 alphanumeric printer prints 40 character lines at 75 lines per minute (Courtesy of Southwest Technical Products Corp., San Antonio, TX)

waste of energy and resources. Even if your application requires some hard copy, your should always be looking for ways to reduce the amount.

One problem in purchasing a printer is the tremendous variety of devices. Note the variations, for example, in Figures 5-14, 5-15, 5-16, 5-17, 5-21, and 5-22. Prices range from a few hundred dollars to many thousands. Other factors include speed, type of paper, interface, method of forming characters, forms capabilities, paper entry methods, reliability, character sets, supplies required, noise level, maintenance methods, graphics capability, and amount of printing that a device can handle. Printer technology is changing rapidly and the user may find a printer more difficult to select than a computer and more expensive as well.

FACTORS IN SELECTING A PRINTER

Among the factors you should consider in selecting a printer are the following:

How it forms characters. Some printers form characters by striking the paper with a character-forming element, much like a typewriter. Typical elements are the ball (used on IBM Selectrics), the daisy wheel (see Figure 5-18) used by Diablo and Qume among others, and the thimble (see Figure 5-19) used in the Spinwriter from NEC Information Systems. These printers produce very clear print (so-called *letter-quality*) that is attractive and easy to read; the characters are said to be *fully-formed.* Other printers (called *dot-matrix printers*) form characters from a set of dots as shown in Figure 5-20. The number of dots varies; 5×7, 7×9, and 9×12 are common. More dots means more distinct and more legible characters.

How it creates marks on the paper. Some printers (the so-called *impact* type) actually strike the paper. Others (*nonimpact* printers) do not strike the paper but use heat, electricity, or jets of ink to make marks. Impact printers use ordinary paper and can form their characters by either the fully-formed or the dot-matrix method. Nonimpact printers generally require special paper and always use dot-matrix techniques. Common, inexpensive nonimpact printers are thermal printers (often used in low-cost printing calculators) that form characters by applying heat to special heat-sensitive paper, and electrostatic printers that form characters by electrically burning the metallic coating from special silvery electrosensitive paper.

What kind of paper it uses. Common options are ordinary, heat-sensitive (for thermal printers), and electrosensitive or aluminized (for electrostatic printers). Obviously, ordinary paper is cheaper, more widely available, more suitable for business uses, and available in more sizes and forms.

How paper is entered. Some printers (see Figures 5-14 and 5-15 for examples) require special rolls of paper, others (Figure 5-17) allow the entry of single sheets like a typewriter, while still others (Figures 5-16 and 5-21) have sprockets that allow you to enter standard computer paper that has corresponding holes and is folded back and forth (so-called *fanfold paper*). The folding is evident in Figure 5-16.

How paper is fed into the printer. The common options are friction feed or pinch roller in which paper is moved by friction between the primary roller and the pinch roller (as in a standard typewriter), and tractor feed in which the paper

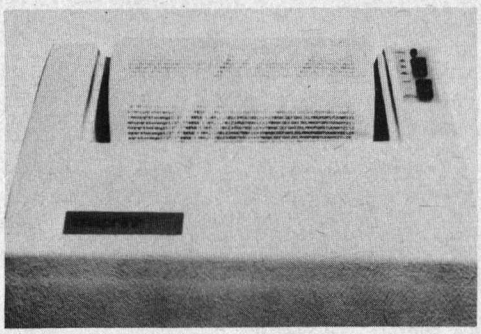


Figure 5-15. The Comprint 912 electrostatic printer prints 80 character lines on 8½-inch paper at 225 characters per second (Courtesy of Computer Printers International, Mountain View, CA)

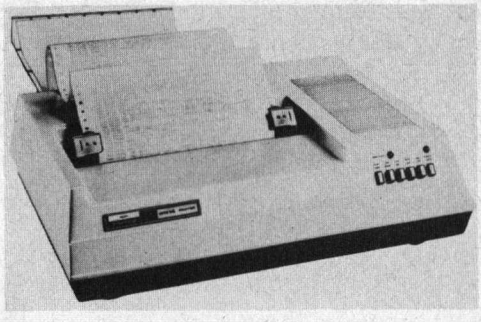


Figure 5-16. The Heath WH-14 printer. It prints 5 × 7 dot-matrix characters at up to 135 characters per second. (Courtesy of Heath Co., Benton Harbor, MI)

is moved by advancing the sprockets that fit into the corresponding holes in the paper. Friction feed mechanisms are cheap, easy to maintain, and tolerant of misalignment but do not produce the precise alignment of print that is required by most forms. Tractor feed mechanisms usually cost more but are essential for precise alignment. Figures 5-16 and 5-21 show variations of the tractor feed mechanism.

Whether it can produce multiple copies. Impact printers can produce multiple copies since they actually strike the paper; fanfold paper can be purchased with carbons included. Thermal and electrostatic printers can only produce one copy at a time. A further problem with thermal printers is that their output does not duplicate well; on the other hand, duplicates of output from electrostatic printers are often clearer than the originals because copiers do not pick up the silvery background.

How fast it prints. Common speeds are 10, 15, 30, 45, 55, 135, and 225 characters per second. In general, dot-matrix printers can be faster than fully formed printers that must rotate their character-forming mechanisms mechanically. Thus the fastest fully-formed printers operate at only 55 characters per second, while dot-matrix printers can be much faster. You should note that Selectric printers can print only 15 characters per second while a standard teletypewriter can print only 10. If we assume that a page contains 2,000 characters, Table 5-1 shows how long it will take printers working at various

Table 5-1. Time required to print a page of text (2,000 characters)

Printer Speed (Characters Per Second)	Time (S)
10	200
15	133
30	67
45	44
55	36
135	15
225	9

speeds to type a page. The teletypewriter and Selectric printer are suited to a human typist, but are quite slow when a computer must print long reports or listings.

Whether it is a character or a line printer. Most inexpensive printers form one character at a time like a typewriter. These are called *character* or *serial printers.* More expensive printers form an entire line at once and are called *line printers.* Because of their high cost, line printers are more commonly used with large computers than with personal computers. There is some confusion here; manufacturers occasionally refer to dot-matrix character printers like those in Figure 5-21 as line printers.

What sizes of paper it allows. Some small printers (such as the one in Figure 5-14) can use only special, narrow paper that is unsuitable for many applications. Most printers (see Figures 5-15, 5-16, and 5-17) allow 8½" or wider paper. Some printers (see Figures 5-21—particularly the models in the front—and 5-22) have wide carriages that can accommodate wide forms. More expensive printers usually have adjustable width like a standard typewriter.

How much printing it can handle. Volume is the key factor here. Most small

"SMART" PRINTER TERMINAL--A microprocessor-based daisywheel printer terminal from Qume Corporation, called the Sprint 5, has been specifically designed for easy integration into data processing applications. The new terminal will bring Qume's well-known high print quality to the computer industry. The Sprint 5 is offered in keyboard and receive-only versions.

5-17. The Qume Sprint 5 daisywheel printer in its keyboard send/receive (KSR) and receive-only (RO) versions. It prints up to 55 characters per second. (Courtesy of ITT Qume, Hayward, CA)

printers can handle 1,000 to 40,000 sheets per month. Larger amounts of printing require a more expensive printer for reasonable reliability.

Which characters it can print. Some printers only produce capital letters; this is clearly unsuitable for word processing applications. Only a few printers have the special characters required for mathematics, foreign languages, or other special applications. With fully-formed printers, the character set depends on the mechanism (note the examples in Figure 5-19).

What kind of interface it has. Some printers have standard serial interfaces, most often RS-232 (see Chapter 6 and Appendix 4). Others have parallel interfaces (see Appendix 4); the most common is called a *Centronics interface* after a leading manufacturer of small printers. Some printers (particularly those available at bargain prices) have no interface; the user must provide the electrical connection to the computer (and the software).

How reliable it is. The more mechanical parts and contacts there are, the less reliable the printer will be. Dot-matrix printers are more reliable than fully-formed printers, and nonimpact printers are more reliable than impact printers.

Whether the character set or type face can be changed. Fully-formed printers (such as the Selectric, daisy-wheel, or thimble types) allow very simple changes. Qume, for example, produces more than fifty different daisy-wheels (Figure 5-18) for its printers. NEC Information Systems also produces many thimbles for its Spinwriter; the one in Figure 5-19 has 121 characters, including fractions, foreign currency symbols, trademark and copyright registers, and mathematical symbols. IBM and other manufacturers offer many type balls for Selectric printers.

Whether it can handle preprinted forms. Only impact printers can handle standard forms such as purchase orders, invoices, registration forms, insurance forms, or tax returns. A tractor feed mechanism is also essential for consistent alignment of printing.

How legible its output is. Fully-formed printers produce far more legible output than do dot-matrix printers, particularly if the number of dots is small.

How noisy it is. Impact printers are far noisier than nonimpact printers. Furthermore, fully-formed printers are usually noisier than dot-matrix printers. Because they operate so quietly, thermal printers are often used in situations (like hospitals) where a low noise level is essential. Teletypewriters, particularly

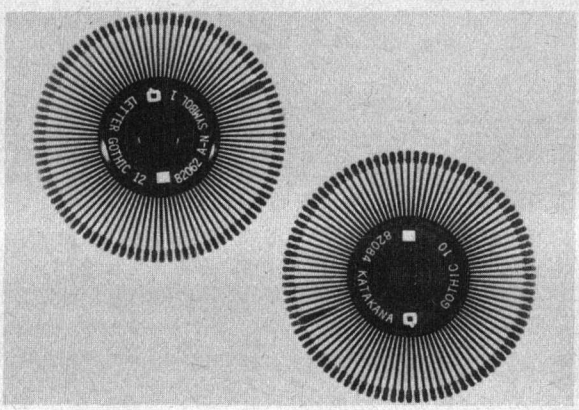


Figure 5-18. Two of the more than fifty different Qume daisywheel print mechanisms (Courtesy of ITT Qume, Hayward, CA)

Figure 5-19. The NEC Super Courier Font with 121 characters including copyright and trademark (Courtesy of NEC Information Systems, Lexington, MA)

the older ones that have many mechanical parts, are very noisy and often require a separate room or a noise-shielding enclosure.

What expendable supplies it requires. Besides paper, some printers require ribbons, ink cartridges, additional character-forming elements, etc.

How easy it is to use and maintain. Some printers are easy to use. Paper can be added easily and preventive maintenance is much like the periodic cleaning of a typewriter. Other printers are quite complex. Paper may have to be entered and aligned precisely and maintenance may be difficult and messy. Misalignment of paper can cause frequent jams that result in lost output and require constant attention.

Whether it has graphics capability. Some printers can generate simple plots and pictures. Graphics are often available as an extracost option in a very limited form. The user should not expect high-quality plots.

Whether it includes a keyboard (i.e., whether it is a complete printing terminal) Some printers always include a keyboard, while others (see Figures 5-17 and 5-22) have alternative models — a KSR (keyboard send/receive) model has a keyboard while an RO (receive-only) model does not. Note the alternatives in Figures 5-17 for the Qume Sprint 5. Printers intended exclusively for use with computers (see Figures 5-14, 5-15, and 5-16) do not have keyboards. If you already have a keyboard, you may prefer a receive-only printer, since the KSR models are bulkier and more expensive than the RO models.

GENERAL CHARACTERISTICS OF PRINTERS

Since selecting a printer involves so many factors, we will now summarize the characteristics of the inexpensive printers and the features that add to the cost.

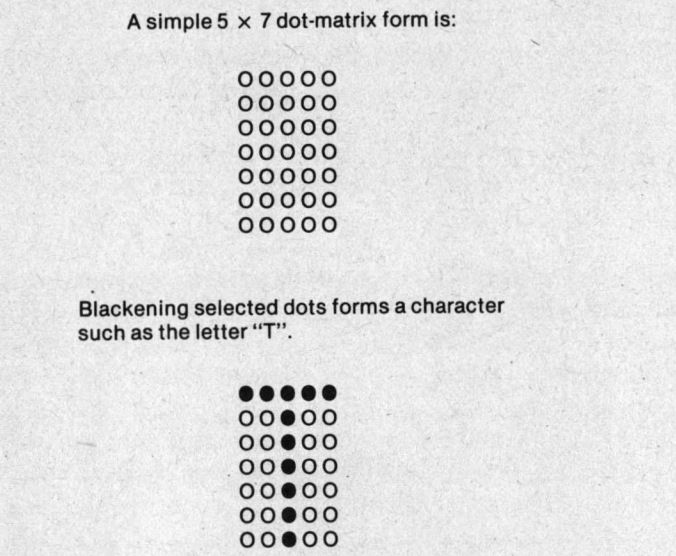

A simple 5 × 7 dot-matrix form is:
Blackening selected dots forms a character such as the letter "T".
More dots make the characters more distinct and less grainy.

Figure 5-20. The dot-matrix method of forming characters

We then discuss the characteristics of different types of printers and the requirements of various applications.

In general, low-cost printers have the following characteristics:

—Low speed, seldom above 30 characters per second
—Low print quality and legibility
—Only suitable for low-volume, light-duty printing
—Provide only a single type font and character set
—Handle only one type and width of paper
—Cannot handle forms effectively
—Lack special features such as graphics, tractor feed, self-test, and error and status reporting
—Noisy and unreliable, if of the impact type
—Often limited to capital letters, small character sets, single copies, and special paper
—Can be used with almost any computer; easy to interface

The characteristics of a printer that one pays extra for are the following:

—High or letter-quality print
—Printing on ordinary paper
—Forms capability, with tractor feed
—Ability to produce multiple copies
—High speed
—Large character sets
—Variations in paper width, print size, character set, or type font
—High reliability
—Ease of use and maintenance
—Quiet operation

—Heavy-duty capabilities
—Graphics or other special features

More expensive printers with more features require more complex interfaces, involving both additional hardware and additional software. As with video displays, extra features on a printer are of no value if the software does not support them.

Let us now look at the characteristics of different types of printers: electrostatic, thermal, impact dot-matrix, and impact fully-formed. Remember that electrostatic and thermal printers always use the dot-matrix method to form characters. We can characterize printers by type as follows:

Thermal printers, like the small one shown in Figure 2-1 on the Rockwell AIM-65, are quiet, reliable, easy to use, and generally inexpensive. On the other hand, they require special paper (which may discolor and does not reproduce well), cannot handle forms or produce multiple copies, provide only moderate speed and fair print quality, and cannot offer interchangeable fonts or character sets.

Electrostatic printers (see Figure 5-15) are quiet, reliable, inexpensive, easy to use and interface, and can provide moderate speed. Some offer extra features such as plotting or special characters. On the other hand, they require special paper, cannot handle forms or provide multiple copies (however, the output reproduces well), usually produce low print quality (the printer in Figure 5-15 is exceptional for its quality among electrostatic printers), and cannot offer interchangeable fonts or character sets.

Impact dot-matrix printers (see Figures 5-14, 5-16, and 5-21) provide moderate to high speed, use ordinary paper, and often allow varied widths, forms handling, and multiple copies. On the other hand, such printers are noisy and only moderately reliable. The print quality is fair to good at best and the output is usually difficult to read under close scrutiny. A tremendous variety of these printers exists; Figure 5-21 shows the models available in just one series from one manufacturer.

Figure 5-21. The Centronics 700 Series of serial printers. Speeds range from 10 characters per second to 370 lines per minute. (Courtesy of Centronics Data Computer Corp., Hudson, NH)

Figure 5-22. The NEC Model 5540 KSR APL Spinwriter thimble printer. The printing rate is up to 55 characters per second. (Courtesy of NEC Information Systems, Lexington, MA)

Impact fully-formed printers (see Figures 5-17 and 5-22) provide the highest print quality. They use ordinary paper with varied widths, have forms capabilities, and allow easy interchange of fonts and character sets. Fully-formed printers also provide multiple copies and are often usable off-line as typewriters. On the other hand, their speed is moderate at best and they are noisy, expensive, relatively unreliable, and difficult to interface. We should note that Selectric printers are suitable only for low-volume applications because of their low speeds and fairly high wear. Local maintenance and prompt service are widely available for the more popular models of the fully-formed type because of their widespread use and strong backing by the manufacturers.

PRINTERS FOR VARIOUS APPLICATIONS

We may generally characterize the type of printer required for various applications as follows:

- An impact, dot-matrix printer is usually best for general business applications because it can handle forms, provide multiple copies, and operate at high speed.
- A fully-formed printer is essential for word processing applications because it is the only type of printer that produces letter-quality output. Daisywheel or thimble printers are superior to Selectric printers because they are faster and more reliable.
- Thermal or electrostatic printers are usually best for home or educational use where high-quality print is not required and the amount of printing is small. These printers are quiet, inexpensive, and reliable. The cost of the special paper is a disadvantage and the purchaser should check its price and availability. These printers are also suitable for laboratories, desktop systems, and business and professional applications where the printed output is only used locally or informally.

We should note that printers, despite their relatively high cost, are typically the most variable, least reliable, and most difficult peripherals to use. Far more can go wrong with a printer (particularly an impact printer) than with a keyboard, video display, cassette recorder, or floppy disk. If you plan to use your printer heavily, you should look for one with a proven record and local service. A maintenance contract is essential, just as with an office typewriter, and you should schedule regular service. We discuss the maintenance of printers further in Chapter 7.

OTHER PERIPHERALS

Other peripherals that are occasionally used with personal computers include the following:

Voice input devices. These allow a computer to recognize a limited vocabulary of precisely spoken words.

Voice output devices (see Figures 5-23 and 5-24). There are two major types. The addressable reconstruction devices (see Figure 5-23) use a small, fixed vocabulary, but produce high-quality speech. Speech synthesizers (see Figure 5-24) allow an almost unlimited vocabulary, but produce the mechanical, uninflected speech so often associated with computers.

Paper tape readers and punches (see Figure 5-25 for a popular, low-cost model) are a throwback to computers of the 1960s. They are slow, noisy, and unreliable, and fortunately have been superseded by cassette recorders and disks.

Analog I/O systems allow the computer to measure and control continuously changing conditions such as temperature or pressure.

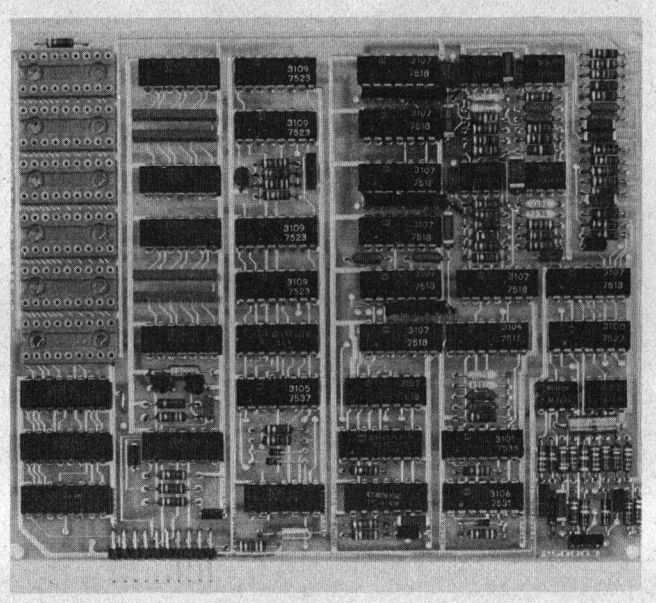


Figure 5-23. The Speech Technology M250 voice generator (Courtesy of Speech Technology Corp., Santa Monica, CA)

Figure 5-24. The TRS-80 voice synthesizer for TRS-80 computers (Courtesy of Radio Shack Division, Tandy Corp., Fort Worth, TX)

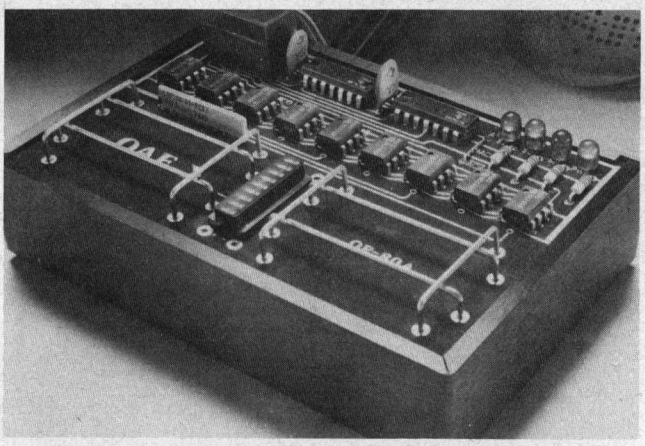


Figure 5-25. The Oliver Advanced Engineering OP-80A optical paper tape reader. Unlike mechanical readers, this device is quiet and reliable since it reads the tape optically; however, the user must pull the tape through the reader. (Courtesy of Oliver Advanced Engineering, Glendale, CA)

Digital control interfaces allow the computer to control lawn sprinklers, lights, furnaces, and other electrical or mechanical systems.

Music synthesizers allow the computer to generate music.

Plotters (see Figure 5-26) allow the computer to draw graphs and figures.

Digitizers (see Figure 5-27) allow you to enter shapes as data.

Communications interfaces and acoustic couplers allow the computer to communicate with other computers over the telephone lines and to obtain information from remote data bases, such as stock market quotations.

Mark-sense card readers can be used to enter data from cards that you can mark with a pencil. These cards are often used in automatic test scoring.

Real-time clocks allow the computer to keep track of actual calendar time.

Camera interfaces and video digitizers allow the computer to handle video input data.

Multiterminal adapters allow one computer to service many terminals. These adapters require special software so the users do not interfere with each other (such software is referred to as *multi-user* for obvious reasons).

The most fascinating of these peripherals are the voice I/O devices since they allow the use of the oldest and most common means of human communications — the voice. Surely no science fiction film can show a computer without assuming that the people of the future can ask it questions and receive answers in an eerie, metallic, uninflected voice. Besides this intrinsic fascination, voice input/output has obvious applications in games, computer-aided instruction, language translation, and even medicine and business. The possibilities for aiding those with visual, hearing, speech, or physical handicaps are endless. We can expect a rapid expansion of the number and capabilities of voice I/O devices.

We should note that a computer can, in theory, use any device as a peripheral that a person can use. The device could be a musical instrument, tool, electrical appliance, valve, switch, or any piece of electrical or mechanical equipment. Of course, the computer can only control a device in practice if the appropriate interface and software exist.

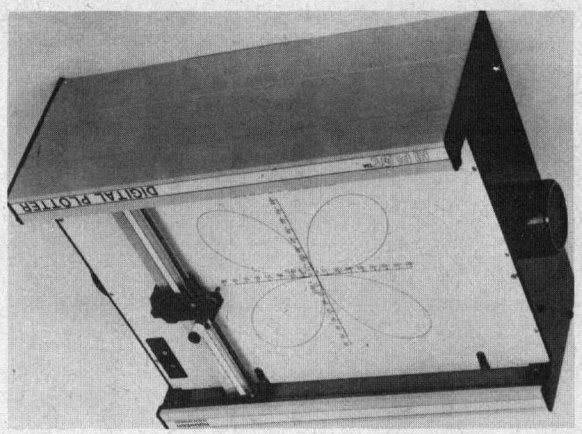


Figure 5-26. The Houston Instrument HiPlot Digital Plotter for personal computers (Courtesy of Houston Instrument Co., Austin, TX)

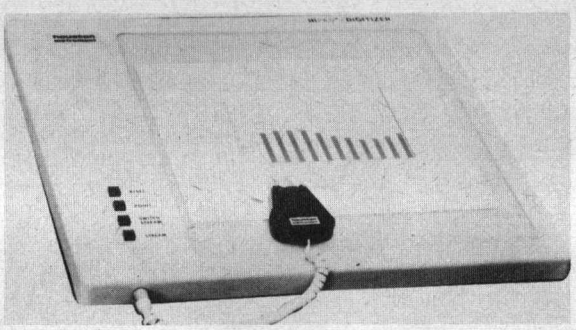


Figure 5-27. The Houston Instrument HiPad Digitizer for graphics input (Courtesy of Houston Instrument Co., Austin, TX)

SUMMARY

Today's personal computers are made useful by the availability of high-speed, reliable, convenient, relatively inexpensive peripherals. Keyboards allow users who can type to enter programs and data quickly. Video displays operate quietly and rapidly, showing the results in a highly legible form without wasting paper. Cassette recorders and disk systems can be used to save programs and data; cassettes and diskettes also provide convenient forms for purchase or exchange. Many different printers can provide a permanent record (hard copy) of listings and results when necessary. Other peripherals have not yet been widely used but offer fascinating possibilities for the future.

One problem is that users find peripherals as difficult to evaluate and select as computers, and often more expensive as well. Manufacturers are constantly producing new peripherals and the number of variations in standard devices is astounding. Most users should buy peripherals that are simple and convenient to use, easy to connect to the computer, and provide the most features. Users should make sure that the peripherals they buy are compatible with their computers and provide the features required for their applications. Those with limited requirements or special skills may find excellent bargains in peripherals that have limited capabilities or are relatively difficult to interface.

6 Interfacing

This chapter discusses interfacing, the connecting of peripherals to a computer. This is a complex topic for the following reasons.

1. The tremendous differences among peripherals: Peripherals operate at different speeds, record data in different ways, and use different media. Consider, for example, how different a printer is from a cassette recorder, a keyboard from a video display, or a typewriter from a floppy disk.

2. The differences among computers (although computers obviously have more in common than do peripherals): Computers run at different speeds, have different instruction sets, and transfer data to or from peripherals in different ways. Interfacing the same peripheral to different computers is like installing the same accessory in different automobile models — the basic idea is always the same but the specific procedures vary greatly.

After programming, interfacing is the most difficult aspect of computers to explain. Beginners often believe that you can plug a peripheral into a computer, much as you plug a household appliance into the nearest electrical outlet. In fact you can only plug a peripheral into a computer if either the computer or the peripheral contains the appropriate interface. Otherwise attaching the peripheral to a computer is like attaching a mechanical arm to a person; the arm may be physically attached with miracle glue but there are no muscles with which to move it or nerves with which to sense what it is doing. The interface acts like the body's muscles and nerves. Note that, besides the physical (or hardware) connection, using an arm also requires learning (or software) that must be programmed into the brain.

The interface must allow the computer to:

Transfer data to or from the peripheral.
Determine the current state or status of the peripheral.

The computer must know, for example, if the peripheral is connected and powered and if it has new input data or has finished handling the most recent output data. Status information is like the information the nerves send to the brain about temperature, pressure, presence of objects, and other conditions affecting the body.

Translate the data into the form the peripheral requires or from the form it produces.

Control how the peripheral operates. The computer may be able to choose the spacing of lines on a printer, the speed of a tape recorder, or the direction of motion for a mechanical robot.

Detect operating problems, such as a printer jamming or running out of paper or a cassette recorder reaching the end of a tape.

Time operations correctly. Peripherals operate at particular rates. A cassette recorder, for example, moves its tape at a particular speed while a typewriter prints characters at a maximum rate determined by how fast its printing mechanism can move.

The interface must provide both the hardware and the software to perform these tasks. This chapter first describes hardware and software interfaces, then discusses standard interfaces and input/output boards, suggests criteria for selecting interfaces, and finally discusses the interfacing requirements of common peripherals and the interfaces that are usually available for particular peripherals and categories of personal computers.

THE HARDWARE INTERFACE

Hardware interfaces (see Figure 6-1 for an example) consist of electrical circuits, mechanical connections, power supplies, connectors, mountings, and cables. An interface may be as complex and as expensive as a CPU or memory board; interfaces for large computers and complex peripherals often cost more than the peripherals. The interface may be located anywhere in the computer — it may be included in the peripheral, it may be part of the CPU or another circuit board, or it may be a separate system.

What does the hardware interface do? In its simplest form, it may do little more than physically connect the computer to the peripheral. All the other functions of an interface must be performed in software. The hardware interface, however, may do much more:

- It may convert data from the form used by the computer into the form used by the peripheral (or vice versa). A typical conversion is the formation of the series of dots for a dot matrix printer (see Figure 5-20) from the ASCII characters.
- It may time the operations of the peripheral. Most peripherals transfer data at a regular rate (such as 10 or 30 characters per second) that is far

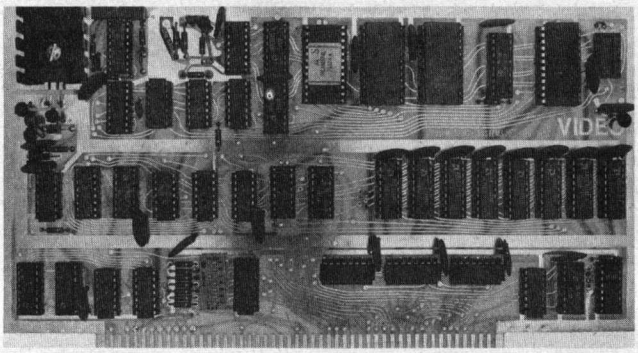


Figure 6-1. The Polymorphic Systems video I/O board (Courtesy of Polymorphic Systems, Goleta, CA)

Figure 6-2. Two models of the Centronics Microprinter—the P1 (left) with a parallel interface and the S1 (right) with an RS-232 interface (Courtesy of Centronics Data Computer Corp., Hudson, NH)

slower than the operating speed of the computer. For example, the computer clearly must not send characters to a typewriter faster than the typewriter can print them; on the other hand, sending characters at a slower rate than the typewriter can handle makes printing slower than it has to be.

—It may have some memory (called a *buffer*) in which data can be saved. The computer can then transfer data to or from the interface in one burst and spend the rest of its time performing other tasks without interruption. Video interface boards (see Figure 6-1) frequently include their own memory, which holds the image that is currently on the screen.

—It may check the peripheral to see whether it is ready to transfer data and is operating correctly. It may watch for problems such as misalignment, lack of paper or tape, failure to record properly, gaps in tape or other media, or loss of data because of timing errors. Interfaces vary greatly in what they monitor and how they report conditions. The variations are like those in the instrument panels of automobiles: expensive cars have many meters and gauges while inexpensive ones have only a few indicator lights. A computer can use status information to operate a peripheral more efficiently, much as a skillful driver can use the information provided by meters and gauges to improve the performance of an automobile.

—It may determine how the peripheral operates by selecting its rate of operation, recording method, spacing of data, direction, and error reporting. Here again, the variations are tremendous and depend on the cost of the interface and the complexity of the peripheral.

Interfaces must be considered in any comparisons among peripherals and computers. Clearly a peripheral that has an interface will cost more than one that does not. Similarly a computer that has interfaces will cost more than one that does not. The two printers in Figure 6-2 differ only in the type of interface they have; despite their similar appearances, the one on the right costs $100 more than the one on the left.

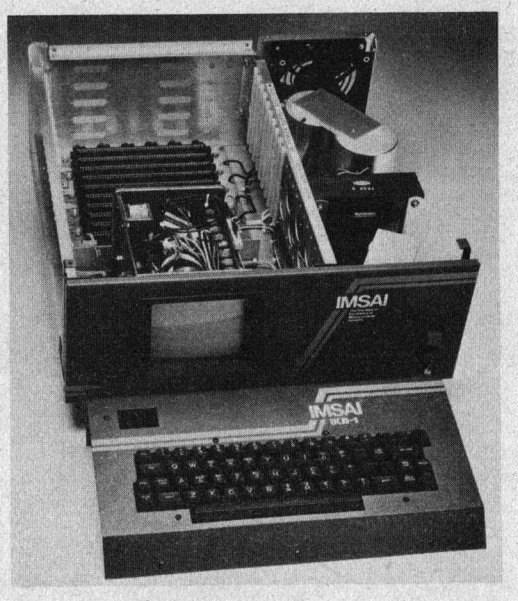


Figure 6-3. The Imsai PCS-80/30 Integrated Video Computer with its cover removed (Courtesy of IMS Manufacturing Co., San Leandro, CA)

An interface also requires a housing, power, connectors, and cables unless it can use those that are included in the computer. Here, again, there are obvious tradeoffs. The computer in Figure 6-3 has slots for interfaces. All the user must do is slide the interface into the slot and connect it. On the other hand, the computer in Figure 6-4 has no place for additional interfaces. You must purchase a separate housing, power supply, and connectors. The computer in Figure 6-4, however, costs less than the one in Figure 6-3 and is a good value if you have no plans for expansion.

Status and control functions provide further tradeoffs. Each function increases the cost and complexity of the computer, just as each extra gauge or dial increases the cost and complexity of an automobile. However, each function also increases the capabilities of the computer. The more the computer can control, the more different operating modes it can provide and the easier it can make the peripheral to use. For example, we have previously discussed the special features (attributes) that a video terminal can provide if there is some way for the computer to control them. The more the computer knows about its peripherals, the more completely and more automatically it can manage them. Of course, each part of the computer must handle status and control functions; there is no sense in the interface reporting conditions that the peripheral does not record or that the software cannot understand. Nor is there any sense in the interface controlling a function that the peripheral does not have or that the software does not support.

Interfaces can thus be a source of confusion. Each peripheral needs an interface, but the interfaces are difficult to evaluate and compare. Clearly you do not want to duplicate interfaces; for example, you would not want to purchase a computer with a video interface, a video interface board, and a video display with its own interface. This would be like buying a stereophonic system that includes speakers when you already have a set of speakers and your house has built-in speakers.

Some peripherals, known as "intelligent" or "smart" peripherals, actually contain their own computers. Since the internal computers can handle many of the details, they make the peripherals easy to interface. Interfacing an intelligent peripheral to a computer is like explaining a difficult job to a trained person; clearly this is much simpler and much less time-consuming than explaining the same job to someone who is completely untrained.

BUILDING INTERFACES

If you can build your own interfaces, you can obviously save money by buying peripherals without interfaces. Such devices are often offered at bargain prices since most people cannot use them. In comparing the prices and capabilities of peripherals, you should always check the cost and availability of the interface, as well as exactly what is needed to make it operate. You do not want to pay for an unnecessary interface, nor do you want to buy a peripheral with no interface (unless you can build your own) or an interface with no place to put it or power to operate it.

Building interfaces is difficult. You should not attempt it unless you have the necessary time, experience, and equipment. Some interfaces are available in kit form, but even these may be surprisingly difficult to build. If you need an interface that is not available, your best bet may be to deal with a company that specializes in custom interfaces.

The newcomer may ask why interfaces are so expensive and so difficult to build. After all, interfaces do not look any more complicated than memory boards or other accessories that are relatively easy to construct. One problem

Figure 6-4. The Radio Shack TRS-80 microcomputer (Courtesy of Radio Shack Division of Tandy Corp., Forth Worth, TX)

with building interfaces is that each one has unique electrical and mechanical requirements. The high cost of interfaces is caused by their long development times and low sales volumes.

Interfaces are also difficult to test, since all they do is connect computers to peripherals. You cannot just turn the interface on and see if it works, as you can with a printer or a tape recorder. You need an oscilloscope or other electronic test equipment to check out an interface like the one in Figure 6-1. If you don't have an oscilloscope or aren't sure of what the output should look like anyway, all you can do is connect the computer to the peripheral and see if the entire system works. If it doesn't work, the problem could be anywhere — in the computer, in the peripheral, or in the interface. Just determining where the problem is could be difficult.

STANDARD HARDWARE INTERFACES

Having each peripheral and each computer act differently is like having each person speak his or her own language. A standard simplifies every aspect of design and development since both computers and peripherals can be equipped with the standard interface, software can be written for it, and systems can be evaluated by cost and performance rather than by how difficult they are to connect.

Many standard interfaces exist; we describe several in Appendix 3. The two most widely used with personal computers are the RS-232 serial interface for peripherals and the S-100 bus for memory boards, CPU boards, and I/O boards. Remember that a bus is simply a set of parallel electrical connections between devices. Other buses include SS-50, Radio Shack, Heath, OSI-48, and IEEE-488 (used in the Commodore PET computer). Some manufacturers make bus converters that connect one bus to another; the most popular converters connect the Radio Shack TRS-80 or Commodore PET (IEEE-488) to the S-100 bus.

RS-232

The RS-232 interface is a popular standard sponsored by the Electronic Industries Association (EIA), which produces many standards for electrical wiring, components, media, and equipment. RS-232 is a serial interface; that is, there is only one line carrying data from one device to another. The alternative is a parallel interface in which there are several data lines. A serial interface is like a one- or two-lane road, while a parallel interface is like a multilane highway or expressway. A serial interface is cheaper to construct than a parallel interface, particularly over a long distance, but it obviously cannot carry as much traffic (data). RS-232 defines the voltage levels for signals, the amount of amplification used, the ways in which devices indicate they are ready to receive or transmit data, and the ways in which devices communicate over the telephone lines. Standard connectors, cables, and electronic circuits are available for RS-232 interfaces.

The RS-232 interface is most commonly used for low- and medium-speed peripherals, such as terminals, printers, and card readers. It is seldom used to transfer data to or from memory or a disk. Extra circuitry must be added to a peripheral to make it follow the RS-232 standard. For this reason, an RS-232 interface may cost extra; the printer in Figure 6-2, for example, costs $100 more with an RS-232 interface.

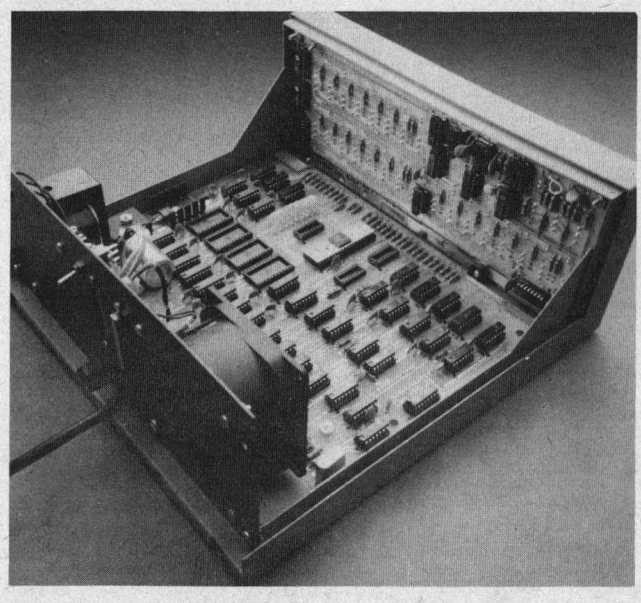


Figure 6-5. An internal view of the MITS Altair 8800A, the first widely distributed personal computer. Its internal connection method became the industry-standard S-100 bus. (Courtesy of Pertec Computer Corp., Los Angeles, CA)

S-100 Bus

Unlike RS-232, the S-100 bus is an accidental standard. Altair Corporation (now part of Pertec), one of the earliest manufacturers of personal computers, developed it in the middle 1970s as a way to connect accessories to the computer shown in Figure 6-5. Other manufacturers copied the bus for use in other computers and in accessories like the one in Figure 6-6. However, no one decided exactly what the standard was until the IEEE (Institute of Electrical and Electronic Engineers) formed a committee to define it. Despite this confusion, the S-100 bus (the "100" refers to the number of wires) is the most popular standard for personal computers.

The S-100 bus is a parallel interface with eight data lines connecting devices, so it can transfer data faster than RS-232 can. As we mentioned earlier, computers that use the S-100 bus generally are based on 8080 or Z80 microprocessors; this is because the original Altair computer (Figure 6-5) was based on the 8080 microprocessor.

The S-100 bus is most commonly used to connect the CPU to memory, disk interfaces, general-purpose I/O boards, and other high-speed peripherals. Figures 6-6 through 6-8 show typical S-100 boards.

THE SOFTWARE INTERFACE

The software interface consists of programs that convert data from one form to another, time the operations, check conditions, and control the peripheral. The hardware interface must provide the mechanical and electrical connections and

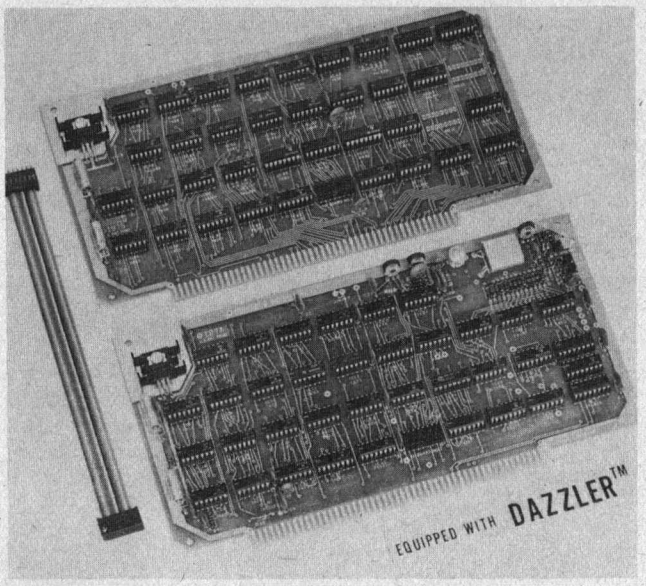


Figure 6-6. The Cromemco TV Dazzler board, used to interface an S-100 bus computer to a color television set. This was one of the first S-100 accessories. (Courtesy of Cromemco, Inc., Mountain View, CA)

local memory such as that used by a video display (remember Figure 6-1). The software must transfer the data to or from the interface. Everything else can be done either by hardware or by software; we discuss the tradeoffs later in this chapter. Like the hardware interface, the software interface may be located anywhere. It may be part of the operating system; most operating systems include drivers for a few common peripherals. Software interfaces may also be separate programs, provided with the peripheral or supplied by the user. These programs are called I/O drivers, I/O utilities, or input/output control systems (IOCS).

Just as with hardware interfaces, systems with different software interfaces may look the same. Obviously, the capabilities of the computer in Figure 6-4 depend on which software interfaces are included in its read-only memory. The capabilities of the computer in Figure 6-3 depend on the kind of software available to handle the various boards that can be inserted. There may be a slot in which to place an interface and enough power to run it, but nothing will happen without software.

Like hardware interfaces, software interfaces are specific to a particular computer (or operating system) and peripheral, and difficult to develop and test. Most are written in assembly language because they involve precise timing and detailed control of electrical and mechanical functions rather than mathematical computations or data processing. The common high-level languages all have difficulties with input and output because they differ so greatly from computer to computer.

Again, like hardware interfaces, software interfaces are easier to develop if the peripheral is intelligent and follows a standard. Unfortunately there are even fewer standards for software interfaces than for hardware interfaces.

An interface involving both hardware and software is even more difficult to test than is the hardware interface by itself. Errors could be in the system software, driver program, computer hardware, interface, or peripheral. All these components may operate correctly by themselves, but may not work together.

The writing of software interfaces generally requires much time and much programming experience. Not only must the programs be written in assembly language, but they must then be connected to programs that are written in high-level languages like BASIC or PASCAL. Thus the availability of software may be a critical factor in evaluating a peripheral or an interface.

HARDWARE/SOFTWARE TRADEOFFS

We have noted that either hardware or software can perform many of the functions of an interface. In general, hardware interfaces are faster since they can perform several tasks at once whereas a computer program does one thing at a time. Hardware interfaces are also easier to change if you can choose data rates and other options with switches, jumper wires, or straps; for example, the interface in Figure 6-7 has six sets of switches for selecting alternative operating modes (four in the upper right-hand corner and two in the lower left-hand area; the white lines are small rocker switches). Furthermore, hardware interfaces can provide their own memory. Extra RAM can serve as local memory that does not interfere with normal computer operations. Extra ROM can customize the interface or contain part of the I/O driver. The interface in Figure 6-7 has slots for optional RAM and ROM.

On the other hand, since they require no additional parts, software interfaces make the computer less expensive and more reliable. Thus software interfaces are cheaper, whereas hardware interfaces are faster and often more flexible. Clearly, manufacturers often make inexpensive peripherals and computers depend on software interfaces to reduce costs.

INPUT/OUTPUT BOARDS

If no specific interface is available for a peripheral, you may be able to adapt a general input/output board to perform the required tasks. Such a board (see Figures 6-7 and 6-8 for examples) is intended to handle simple peripherals such as switches, keyboards, status lights, and terminals. The user generally only has to set switches or add a few wires to meet the needs of a particular peripheral. Of course, any of these I/O boards requires, like any other interface, a housing, power, connectors and cables.

Input/output boards may contain any combination of the following:

—Parallel (8-bit) ports. Remember, a port is the basic unit of the input/output section.
—Serial (1-bit) ports
—RS-232 interfaces
—Standard teletypewriter interfaces

CHOOSING AN INTERFACE

In selecting an interface, you must ask the following questions: With which computers can it be used? What peripherals can it handle? What is required to make it operate? Is the necessary software available? If not, how difficult is it to write? How fast can it transfer data? A key point here is whether it can operate at the maximum rate that the peripheral and software permit. If it cannot, it will slow the entire computer. Does it follow a standard? It will be easier to use a standard interface with a new peripheral or a new computer. Does it support all the functions that the peripheral has or that the software allows? Additional functions allow later upgrading. Where can it be placed? Does it include its own housing, connectors, and power? Does it provide for expansion? Can it handle more than one peripheral? Will it work with your software and your other peripherals? Is it a quality product? Does it have a reasonable warranty and has it been tested? What options does it have and how are they selected?

INTERFACES FOR VARIOUS PERIPHERALS

We now discuss the interfaces commonly available for particular peripherals, their options, and their advantages and disadvantages. The simpler peripherals usually have simple interfaces, often based on RS-232. More complex peripherals, such as disks, can be interfaced in many ways, ranging from slow serial interfaces to complex interfaces that include their own computers.

Interfacing Keyboards

Keyboard interfaces vary according to the size and complexity of the keyboards. Large, stand-alone keyboards are normally interfaced using RS-232. Smaller, simpler keyboards use parallel and serial ports such as those provided by the I/O boards of Figures 6-7 and 6-8. Keyboard interfaces are seldom complex because keyboards usually have few special functions. There are, unfortunately, no standards for options such as upper- or lowercase, special keys, and status and control signals.

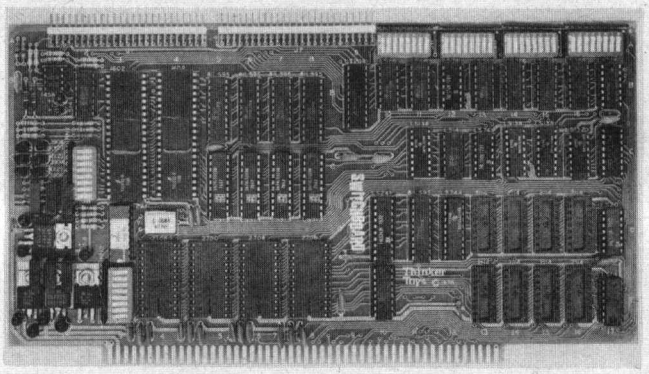


Figure 6-7. The Thinker Toys Switchboard, an S-100 board that provides ROM, RAM, serial I/O, and parallel I/O (Courtesy of Thinker Toys, Berkeley, CA)

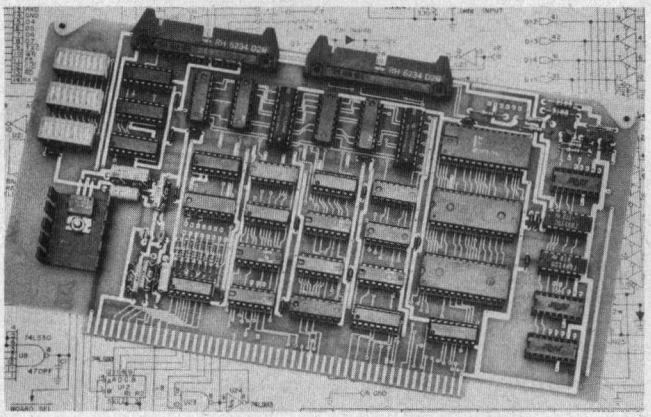


Figure 6-8. The Vector Graphic Bitstreamer II I/O board has two parallel I/O ports and three serial I/O ports (Courtesy of Vector Graphic Inc., Westlake Village, CA)

Interfacing Video Displays

The basic problem in interfacing a video display to a computer is the conversion of computer output signals to the video signals required by a monitor. This is normally performed by a video board with its own memory (see Figure 6-1). Some video boards convert the signals into the form used by television sets. There are also conversion kits (see Figure 6-9) that allow the use of television sets as low-cost video monitors.

Interfacing Terminals

A terminal can, of course, be constructed from a keyboard and a video display. Some manufacturers provide modules (see Figure 6-10 for an example) that make it easy to build home-brew terminals. Obviously the final product will be

Figure 6-9. The Pickles and Trout TV Mod Kit converts a standard television set into a video monitor (Courtesy of Pickles and Trout, Goleta, CA)

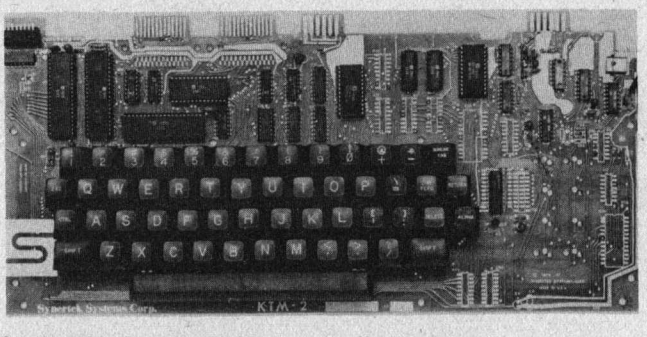


Figure 6-10. The Synertek Systems KTM-2 keyboard/terminal module. The user must add a video monitor. (Courtesy of Synertek Systems, Santa Clara, CA)

inexpensive but of lower quality than a commercial terminal. Commercial terminals usually have an RS-232 interface; a common variable is the maximum rate at which the terminal can transfer data. Higher rates increase the speed at which large amounts of data can be placed on the screen if the computer can provide the data fast enough.

Interfacing Cassettes

There are many ways to interface cassettes to computers. These vary in the way in which the tape is marked for different data values (or *encoding method*), the rate at which data is transferred, the error-detecting and correcting method, and the amount and type of motor control and status reporting that is implemented. Cassette interfaces are always serial since each bit is recorded separately.

Widely used cassette interfaces include Tarbell (see Figure 6-11), Kansas City, PET, and Radio Shack (see Appendix 5 for brief descriptions). These are all mutually incompatible, so that tapes recorded using one interface cannot be played on a computer that uses another.

Interfacing Printers

Printer interfaces also vary greatly. Among the variations are the division between hardware and software, whether the interface is serial or parallel, data rate, error-detecting and correcting method, and amount and type of control and status reporting that is implemented.

The complexity of the interface depends on the complexity of the printer. A simple printer may have a single status line that tells the computer whether the printer is ready. A more complex printer may have other status inputs that report the current position of the printing mechanism, its rate of operation, whether any improper characters have been received, or whether any operating errors have occurred. It will also have many operating modes that the computer must control. If the interface does not handle these status and control functions, the user must either set them manually or else not use some of the capabilities of the printer.

Like other peripherals, the printer may include an interface. The printer may

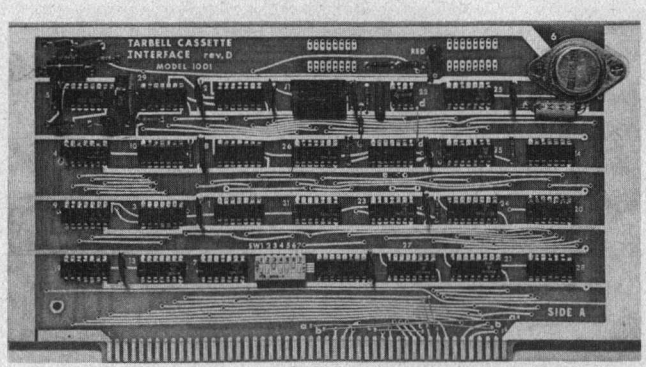


Figure 6-11. The Tarbell cassette interface for the S-100 bus (Courtesy of Tarbell Electronics, Carson, CA)

also have its own computer, allowing it to operate through a simple interface, or its own memory (a buffer), allowing it to receive a large amount of data from the computer at one time. As you might expect, a printer that has an interface, its own computer, or its own memory will cost more than one that lacks such extras. The cheapest printers have no interfaces at all, but they are only useful if you can build or purchase an interface.

Some printers come with a choice of interfaces (remember the example in Figure 6-2). An RS-232 interface allows direct attachment of the printer to most computers. The common parallel interface is the Centronics interface (see Appendix 4); it is generally available on computers and terminals that are advertised as supporting a printer or having a parallel printer port.

Interfacing Floppy Disks

Floppy disk interfaces (and the less common but even more complex hard disk interfaces) vary more than interfaces for any other common peripheral. Variations include division between hardware and software, data rate, error-detecting and correcting method, and amount and type of control and status reporting that is implemented.

Low-speed interfaces operate through regular I/O ports. Popular variations include the North Star and Tarbell interfaces (see Figures 6-12 and 6-13). These interfaces are included in many packaged computers and are often assumed by popular operating systems such as CP/M.

High-speed interfaces transfer data directly between the disk and the memory without intervention by the CPU. This transfer method is referred to as *direct memory access* or *DMA*. High-speed interfaces (see Figure 6-14) usually contain their own computers and will respond to commands issued by the CPU. Computers that use such interfaces can transfer data to or from the disk at high speed but may not be able to run popular operating systems like CP/M without extensive modification.

As with printers, floppy disks are often available at bargain prices with no interface. Since floppy disk interfaces are complex, such bargains are of little value to anyone except the most enthusiastic and determined do-it-yourselfer.

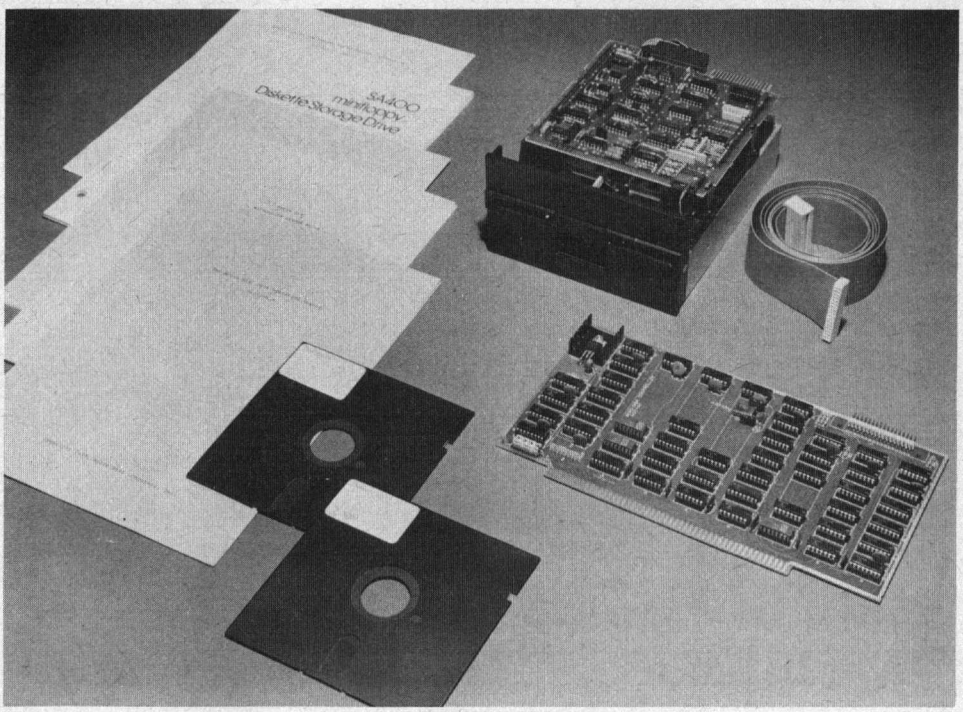


Figure 6-12. The North Star floppy disk system (Courtesy of North Star Computers, Berkeley, CA)

INTERFACES FOR ONE-BOARD COMPUTERS

We now discuss which interfaces are commonly available for the major types of personal computers: one-board computers, appliance computers, and modular computers. We describe how much interfaces cost and how they can be added to these computers.

Most one-board computers include the hardware and software interfaces for an RS-232 terminal, a standard teletypewriter, and an audio cassette recorder, besides the on-board peripherals (such as a keyboard and display). However, there are tremendous variations in how many additional parts and programs are required, how flexible the interfaces are, how fully they implement standards, and whether all peripherals can be attached at the same time. The mere presence of an interface may mean little. Manufacturers generally provide the simplest, cheapest, and least flexible interfaces in order to minimize costs.

Some additional interfaces are available, most commonly for popular computers like the MOS Technology KIM and the RCA COSMAC VIP. Among the offerings are:

—Video display interface (standard on the COSMAC VIP)
—Color television or monitor interface
—Graphics interface
—Tone generator
—Music synthesizer
—PROM programmer

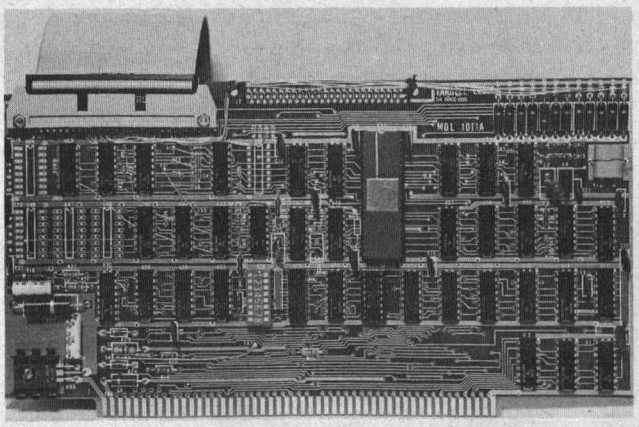


Figure 6-13. The Tarbell S-100 floppy disk interface (Courtesy of Tarbell Electronics, Carson, CA)

—Joysticks
—S-100 bus
—Floppy disk interface
—Simple printer interface

Cost is a major factor here. The cost of the interface and the peripheral can greatly exceed the cost of the one-board computer itself. Obviously it makes little sense to add a hard disk, an expensive word processing printer, or a complex terminal to a one-board computer that only costs a few hundred dollars. This is like trying to pull a large, expensive trailer with a subcompact car; if you can afford the trailer, presumably you can also afford a more powerful car to pull it.

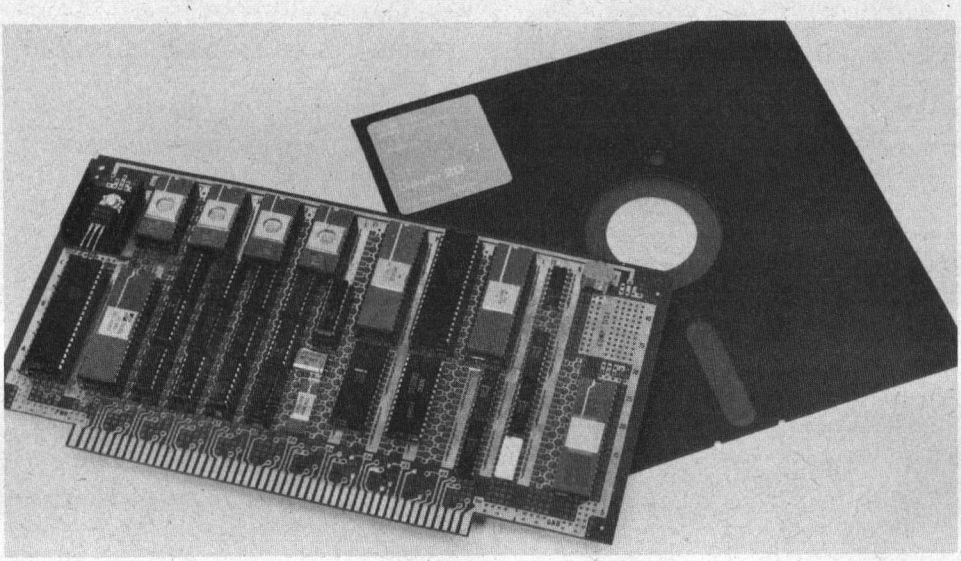


Figure 6-14. The PerSci Model 1170 Z80-based double density diskette drive controller (Courtesy of PerSci, Inc., West Los Angeles, CA)

Even simple interfaces can cost as much as the basic one-board computer. Furthermore, the user must still provide packaging, software, power, connectors, and cables. Expansion modules, like the one in Figure 6-15, can hold extra interfaces and memory. Adding interfaces to a one-board computer allows the user to build a system gradually, but usually results in lower quality than a packaged computer at a higher total cost.

INTERFACES FOR APPLIANCE COMPUTERS

Most appliance computers include the interfaces for a television set or video monitor, a typewriter keyboard, and an audio cassette recorder. As with one-board computers, these interfaces generally use the simplest, cheapest, and least flexible approach in order to minimize the cost of an entry-level system. Each appliance computer has its own interfaces and they are generally not interchangeable. Even the cassette interfaces use different data rates and recording techniques so tapes will only play on one brand of computer.

Because of the popularity of appliance computers, the original manufacturers and other suppliers offer many interfaces. Of course, the variety is largest for the most popular computers — the Radio Shack TRS-80, Commodore PET, and Apple II. Offerings include

—Floppy disk interfaces
—RS-232 interfaces

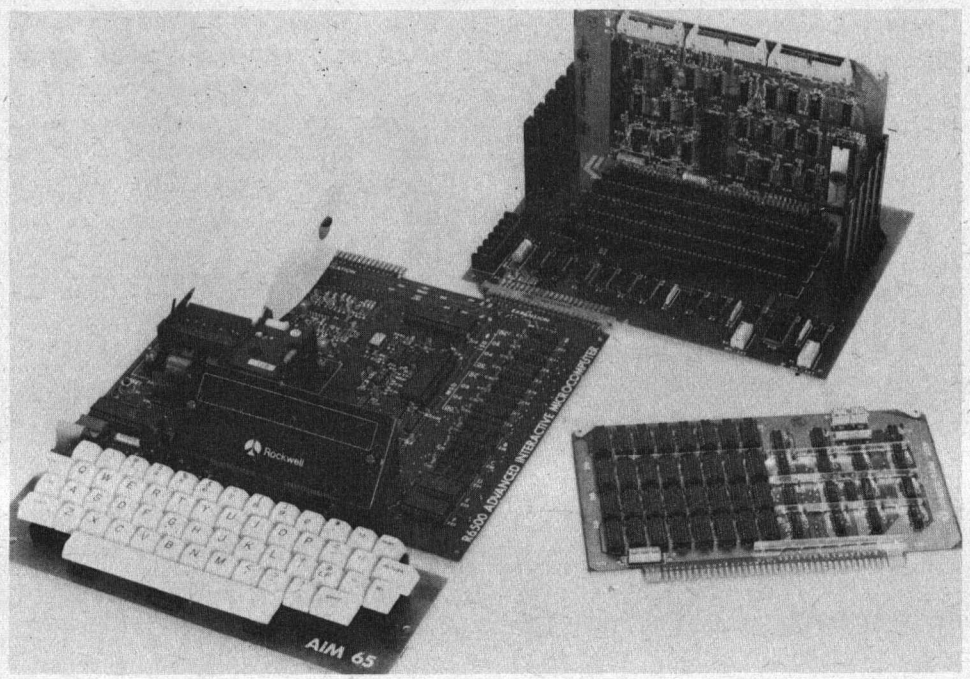


Figure 6-15. The Rockwell AIM-65 microcomputer with expansion module (Courtesy of Electronic Devices Division, Rockwell International, Anaheim, CA)

—Printer adapters for printers ranging from small, simple ones to Selectrics and word processing printers
—Graphics interfaces
—Instrumentation interfaces
—Analog interfaces
—Music synthesizer interface
—Remote digital controllers for lights, switches, and appliances
—S-100 bus interface
—Communications interfaces
—Hard disk interfaces

These interfaces may not be easy to attach. The user generally must provide software, packaging, power, connectors, and cables much as with the one-board computers. Adding interfaces is more difficult with Radio Shack or Exidy computers, which lack space for expansion, than it is with Apple or Commodore computers, which provide standard cases. At best, adding interfaces to an appliance computer often results in a jerry-built system that is difficult to use but costs as much as a well-packaged computer.

INTERFACES FOR MODULAR COMPUTERS

As we have noted in Chapter 2, a modular computer may include no interfaces at all in its simplest form. However, most users purchase a packaged computer which generally includes an RS-232 interface for a terminal, a floppy disk interface, and a printer interface.

These interfaces vary greatly in quality and capabilities. Some are just the cheap, low-quality one-board interfaces in fancier and more expensive packages. Others use the latest technology and offer many options. The cheap interfaces generally allow only a single, low data rate. They often interfere with the expansion of the memory and the I/O section. The better interfaces handle a variety of peripherals and provide higher data rates; they also are designed to permit expansion.

Many other interfaces are available for modular computers, particularly for the S-100 bus. Interfaces are available for peripherals such as:

—Word processing printers
—Hard disks
—Audio and digital cassette recorders
—Card readers
—Cartridge tape
—Voice input and output devices
—High-voltage inputs and outputs
—Prototyping boards
—Sound generators
—Television cameras
—Video digitizers for analysis of video data
—Plotters

These interfaces (see Figures 6-1, 6-6, 6-7, 6-8, 6-11, 6-12, 6-13, and 6-14 for examples) vary greatly in quality and complexity.

Modular computers are the easiest computers to expand. You can insert additional interfaces into slots as long as there is space, enough power, and enough circulation to keep the computer cool. If your application involves several complex peripherals, a modular computer may be cheaper in the long run because of its higher quality, better packaging, and easier expansion, even though it costs more initially than an appliance computer.

SUMMARY

This chapter has dealt with the difficult subject of interfacing. An interface is required to connect a peripheral to a computer; it provides the means by which the two communicate. The design and use of interfaces is complicated by the wide variety of computers and peripherals.

Interfaces are usually difficult to design and test, and depend on the features of a specific computer and a specific peripheral. Interfaces may involve both hardware and software; performing functions in software is cheaper but slower and less flexible. Few users build their own interfaces or write their own software drivers because of the precise timing and detailed control of electrical and mechanical functions that is usually involved.

Some standard interfaces are available. The most popular are the serial RS-232 interface for terminals, printers, keyboards, and other low-speed devices, and the S-100 bus for memory, input/output boards, and disk interfaces. Other buses also exist, but fewer peripherals and circuit boards are built for them. Some simple interfaces can be constructed easily from general-purpose input/output boards.

Of the common peripherals, the keyboards, terminals, and printers generally use either RS-232 or a simple parallel interface based on a few I/O ports. Cassette and disk interfaces do not conform to a single standard. Instead, several incompatible types are widely available. The important trends in interfacing are wider use of standards and more intelligent peripherals (with their own computers and memory) that are easier to connect.

The availability of interfaces and the ease with which they can be added are important factors in choosing among the different categories of personal computers. One-board computers generally have only slow, inflexible interfaces for a few low-speed peripherals such as cassette recorders and simple terminals. Appliance computers also have limited interfaces, although theirs generally operate at higher speed and can handle more substantial peripherals than can those supplied with one-board computers. Modular computers allow the user to add a variety of interfaces, most commonly based on the S-100 bus.

Interfacing peripherals to inexpensive computers is often difficult. The computers may lack a place to put the interfaces, the power to run them, and the software to use them. More expensive computers are easier to expand because they have slots for additional circuit boards, interfaces that can be upgraded, and software that either supports additional peripherals or can easily be modified to handle them.

7 Computer Operation and Maintenance

To most people, operating and maintaining computers is a subject as far removed from everyday life as operating and maintaining nuclear reactors, oil refineries, or spacecraft. People expect computers to occupy their own rooms (as in Figure 7-1) and to require air conditioning, special electrical systems, and highly trained operators.

Today's personal computers differ greatly from that popular conception. They are expected to operate in everyday settings (see Figure 7-2 for an example) with a minimum of special attention. A personal computer (note the collection of parts in Figure 7-3) is no more complex than an office typewriter, calculator, or copying machine.

This chapter describes how to operate and maintain a personal computer. We describe the required surroundings and operating methods and the care

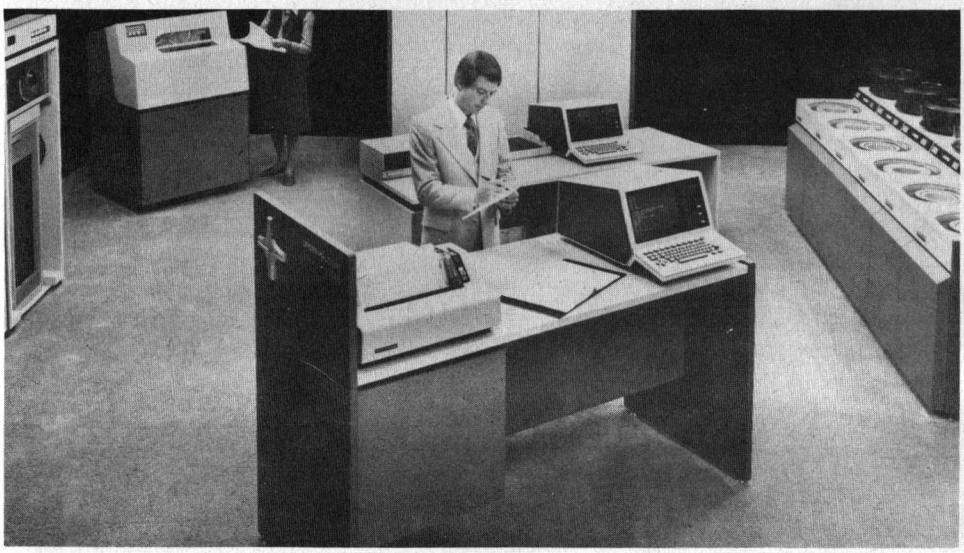


Figure 7-1. The Sperry Univac 1100/60 computer. This is a large mainframe capable of handling enormous data processing tasks. (Courtesy of Sperry Univac, Blue Bell, PA)

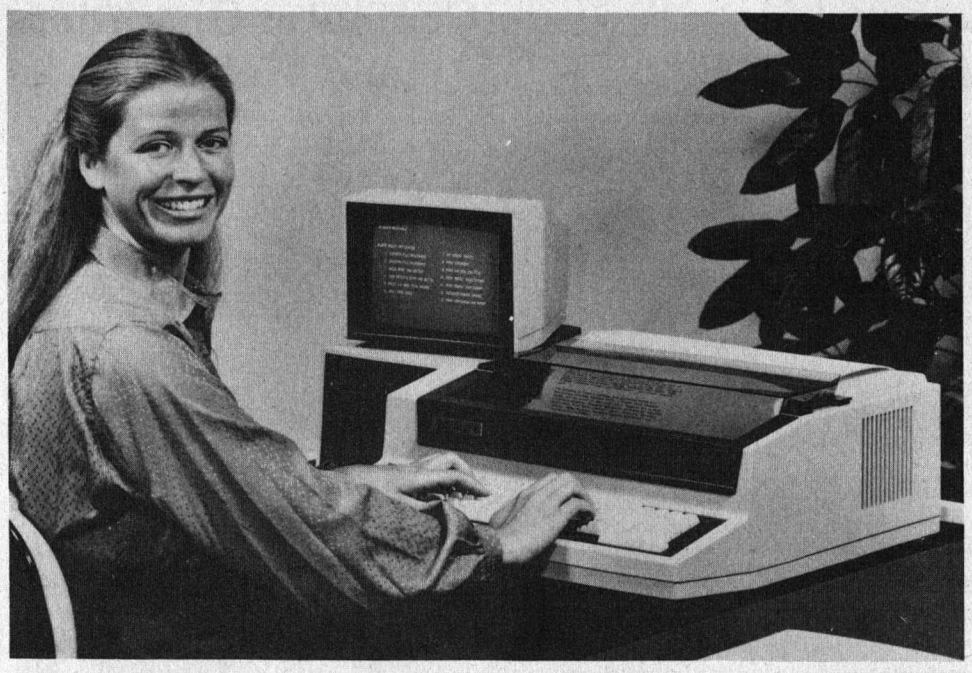


Figure 7-2. The Durango F-85 desktop data processing system. The entire system (including the video display) weighs 65 pounds (30 kilograms). (Courtesy of Durango Systems, Inc., Cupertino, CA)

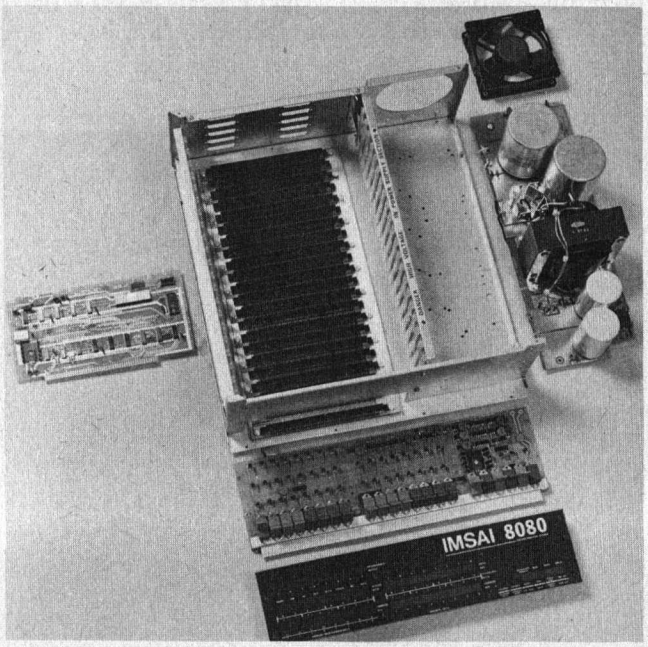


Figure 7-3. The Imsai 8080 microcomputer before final assembly (Courtesy of IMS Manufacturing Co., San Leandro, CA)

and maintenance of the computer, its peripherals, and its software. We also discuss how to recognize problems and pinpoint their sources, what to do if you cannot correct the problem, and what kinds of tools and background you will need if you intend to maintain a computer by yourself. We present operations from the point of view of the average user; we do not describe how to actually construct or repair a computer.

As with most electrical equipment, reasonable care can make a computer last longer and work better. Proper planning can help you avoid minor problems or recover from them with a minimum amount of effort.

THE OPERATING ENVIRONMENT

The first topic we consider is the computer's surroundings or operating environment. The location should be as clean and quiet as possible. Like a television set, a computer works best when it is reasonably cool and not too humid or too dry. Sources of electrical noise like vacuum cleaners, power tools, and floor polishers affect computers just as they affect televisions and radios.

A suitable environment will have the following characteristics:

- Reasonable temperature and humidity. Temperatures of 15 °C (60 °F) to 30 °C (85 °F) and a humidity of 40% to 70% are optimal. The inside of the computer must not become excessively hot; packaged computers usually have fans (Figure 7-3 shows one sitting at the back and Figure 7-4 shows one mounted).
- Little dust or dirt. You can buy dust covers like those used with office typewriters.
- A dedicated power line directly connected to the fuse box with no other outlets. This cuts down on electrical noise. The power outlet should be rated high enough (10 to 20 amperes) to handle easily the computer and its

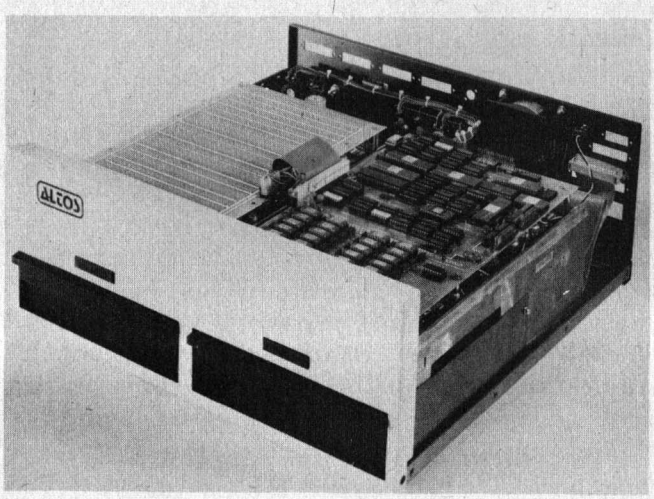


Figure 7-4. The Altos Computer Systems Sun-Series 8000 (Courtesy of Altos Computer Systems, San Jose, CA)

peripherals. Be careful of the peripherals; some, particularly large printers, may require more power than the computer.

- A lightly loaded electrical system. Overloading of the electrical system is often a problem in hospitals and factories that have more equipment than their designers expected.

Overall, we suggest the following guidelines.

—If people find the environment reasonable, the computer will probably find it adequate as well.

—If you can operate an electric typewriter, a stereophonic system, or a television set without any problems, then you can probably operate a personal computer in the same location.

—Don't expect a computer to perform well under conditions such as excessive heat or cold, very high or very low humidity, inadequate power, dirt or other contamination, or excessive electrical noise.

Some things that you can buy to improve the workplace (short of rewiring or adding air conditioning) are dust covers, line filters (to keep out electrical noise), constant voltage transformers, and options like a larger fan, special housing, or better power supply. An electrician can provide you with a dedicated power line at reasonable cost.

INSTALLING THE COMPUTER

Installing a computer requires the same precautions as installing any expensive piece of equipment. You should arrange the system so it can be operated conveniently, moved if necessary, and accessed for maintenance and repair. You should avoid clutter, loose or unmarked cables, and crowding. Figures 7-2, 7-5, and 7-6 show typical arrangements.

Here are some more suggestions for simplifying installation.

Save boxes and other packing materials since you may need them for shipping or to qualify for repair or warranty service.

Arrange all the equipment into a convenient workstation. Try the arrangement and be sure that you can reach everything. Provide easily accessible storage for manuals, programs, listings, and supplies such as cassettes, diskettes, paper, and ribbons. You may want to buy a special desk like the one in Figure 7-6.

Isolate noisy peripherals like printers. For example, the printer in Figure 7-6 has its own stand. Noise shields, padding, or enclosures may help with particularly noisy peripherals such as standard teletypewriters.

Provide easy access to controls, connectors, and cables without having to move equipment. A computer, terminal, or printer generally weighs 10 to 60 pounds (4 to 25 kilograms) so one does not move them casually.

Provide a reasonable writing surface (as shown in Figures 7-2 and 7-6) for making notes, preparing labels, and marking output. Do not use the keyboard, CRT, or other peripherals as writing surfaces.

Leave room for expansion without crowding things together. Plan for likely additions, such as another floppy disk or a printer. Leave room to save disk covers, cassette boxes, and other incidentals.

Figure 7-5. The Radio Shack TRS-80 microcomputer being used in a parts supply house (Courtesy of Radio Shack Division, Tandy Corp., Fort Worth, TX)

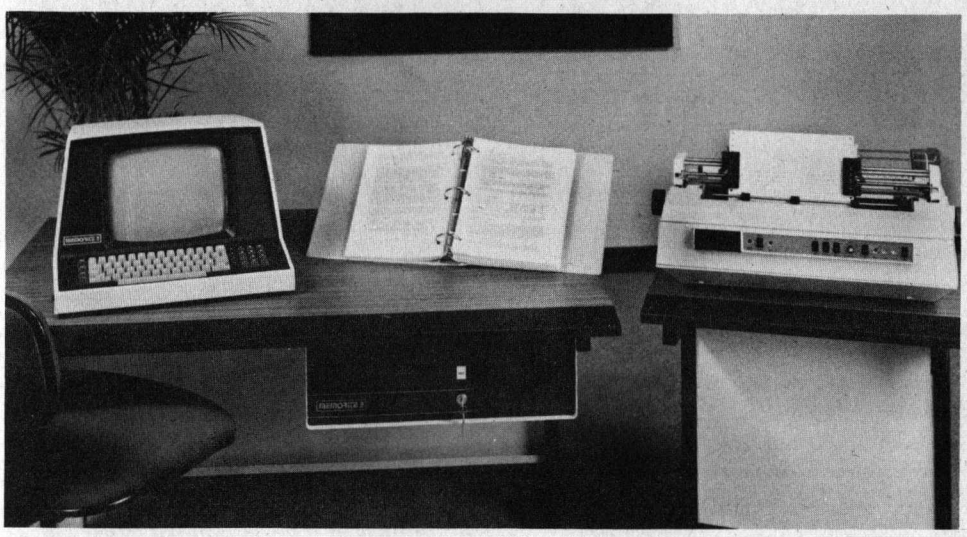


Figure 7-6. The Vector Graphic Memorite 2 Word Processing System with dual minifloppy drives and a Qume daisywheel printer (Courtesy of Vector Graphic, Inc., Westlake Village, CA)

Identify all cable ends and connectors with labeling tape, so you can easily reassemble the system if it is accidentally disconnected or if it must be moved or rearranged. You should also mark the tops of connectors when it is not obvious which side goes up.

Mark all programs and manuals clearly.

Fully assemble all cables and covers, even if you may not use them at first.

Do not stack things on top of one another unless they are specifically designed for stacking. Stacking may block ventilation and cause equipment to become excessively hot.

The key point is to try your arrangement before you make any irrevocable decisions. A trial will show whether the arrangment is convenient and efficient.

INITIAL OPERATION

Before you run the computer, scan the manuals once. Do not expect to understand everything. Only a few manuals (like the *TRS-80 Level I User's Manual* in Figure 7-7) are specifically intended for beginners. What you want is an overview of how to operate the computer—which are the most important controls and how they work, how the manual is organized, and what you should check before starting the computer. Watch for warnings and notes that are derived from the experiences of earlier users. If possible, go over the startup procedure with your dealer or with an experienced user.

You should learn how to operate each element of your system, one at a time if possible. You may want to start without the peripherals and add them later. Important things to learn about using the computer and each peripheral are

Figure 7-7. The Radio Shack TRS-80 Level 1 microcomputer. The manual is specifically written for beginners. (Courtesy of Radio Shack Division, Tandy Corp., Fort Worth, TX)

—How do you turn it on and off?
—How do you stop it or reset it if you are not sure of what it is doing?
—Is the order in which you do things critical? Often you must start the computer before the peripherals or vice versa. Sometimes you must hold one switch in a position while you move others.
—How do you load a simple program?
—How do you load and use systems software such as operating systems, editors, assemblers, and debuggers? Some of these may be readily available in read-only memory while others may require elaborate loading procedures.
—How do you load and use applications programs?
—How do you write a simple program of your own? How do you edit, assemble or compile, load, and execute it?
—How do you save a program or data so you do not have to enter it repeatedly?

One problem is that most computer manuals are intended as references rather than as textbooks. A few manufacturers (see Figures 7-7 and 7-8) produce manuals aimed at the beginner. Most other manufacturers stress completeness. Their manuals describe every feature of the computer, but typically leave it to the reader to extract the information he or she needs. Such manuals seldom contain startup instructions, brief descriptions of the most

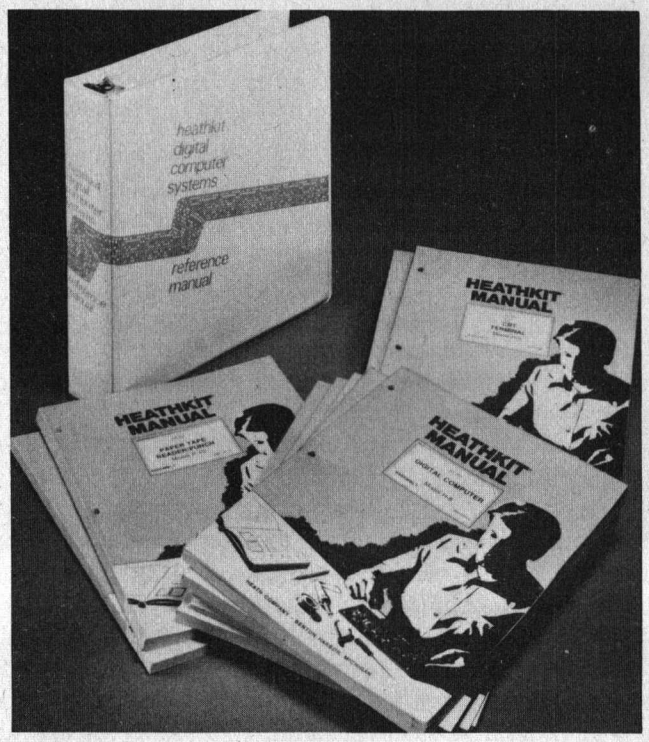


Figure 7-8. The documentation supplied with the Heath H-8 computer (Courtesy of Heath Co., Benton Harbor, MI)

common procedures and features, meaningful examples, or explanations of the reasons behind things. Like most reference books, such manuals make dull reading at best. The beginner should not be discouraged if he or she can learn little from them.

COMPUTER MAINTENANCE

The computer itself is usually the most reliable part of your system. Most computers have few controls and no mechanical components other than a few switches and a fan. The lifetime of the electrical components is so long (500 years or so) that you do not have to worry about them; there really isn't much that can go wrong.

About the only precautions you should take in dealing with the computer itself are these:

- —Be sure that the ventilation is not blocked. Otherwise, the computer may malfunction because of excessive heat.
- —Be sure that all cables are connected properly and identified.
- —Be sure that all controls are accessible and clearly marked.
- —Never remove connectors, covers, or circuit boards while the system is powered. Unplug the computer before removing anything.
- —Keep static charges away from the computer. Such charges are particularly common in dry climates and in carpeted rooms. To reduce their frequency, place the computer in an uncarpeted room; if necessary, you can buy a humidifier or an antistatic floor mat.

CASSETTE MAINTENANCE

Maintaining a cassette recorder used with a computer (as in Figure 7-7) is almost the same as maintaining one used with a home entertainment system. You should do the following.

- —Clean the recorder heads as required, typically after every ten hours of use. Cotton swabs and isopropyl alcohol will do the job.
- —Use short tapes. Many suppliers offer 5- to 15-minute versions. Longer tapes increase searching times, speed variations, and breakage problems.
- —Use high-quality tapes. Some manufacturers offer special data or digital cassettes like the ones in Figure 7-9; these cost more than regular audio tapes but they are more reliable and capable of higher speed. Avoid cheap audio tapes and chromium dioxide tapes that wear down recorder heads.
- —Keep cassettes in dust-proof plastic boxes when not in use. The boxes are available at low cost in most radio/electronics supply stores.
- —Note the operating levels and volume settings required to read each tape properly. Different tapes often require different settings; you should mark the setting on the tape label.

Figure 7-9. Media intended for use with computers (disks, tapes, and cards) (Courtesy of Verbatim Corp., Sunnyvale, CA)

—Use enough leader to avoid difficulties in picking up the start of the recording. Be careful not to record on a plastic leader.
—Identify all cassettes with labels. Packages of labels are available in office supply stores and stereo equipment stores.
—Duplicate tapes as soon as possible. Never depend on a single recording or a single tape, particularly if you use a program frequently.
—Use a bulk eraser (available at stereo supply stores) to completely clean any tape that is no longer in use.
—Demagnetize the tape heads if needed with a degausser (these are also available at stereo stores for a few dollars).
—Use the write protect tab to protect tapes that you do not want to change accidentally.
—Repair or replace cassette housings if they are damaged.

You should *not* do the following.

—Leave the recorder in "play" or "record" when it is not in use.
—Stack cassettes. The housings can warp.
—Use "fast forward" or "reverse" ("rewind") without stopping the recorder first. Some units will not allow this and it can strain the mechanism and stretch the tape.
—Place cassettes near the magnetic fields produced by transformers, power supplies, speakers, or motors. Do not put cassettes on top of the cassette recorder since it contains a motor.
—Allow cassettes (or the recorder) to become hot or contaminated with dirt, water, or other foreign substances.
—Repair breaks in cassette tapes with cellophane tape; use only proper splicing tape.
—Try to record on a tape without removing the write protect tab.

FLOPPY DISK MAINTENANCE

Maintaining floppy disks requires many of the same precautions as maintaining cassettes or phonograph records. Practices to follow include

—Keep the door on the disk drive closed when the unit is not in use. This keeps out dust.
—Keep diskettes in their boxes (see Figures 7-9 and 7-10) or in holders that keep them upright. Otherwise, diskettes may warp. Diskette holders are available in computer stores and from mail-order suppliers.
—Identify all diskettes with labels as shown in Figure 7-10.
—Turn the disk drive off (if this is required) before changing diskettes.
—Insert diskettes correctly, noting the markings on the cover.
—Duplicate all diskettes for backup.
—Use the write protect tab if you don't want data or programs on the diskette to be changed accidentally.
—Store the diskettes in a cool, clean place.

Practices to *avoid* include

—Writing on the disk cover. Write on the sleeve with the disk removed or on labels.
—Leaving diskettes lying around unprotected.
—Touching exposed portions of the disk; only touch the paper jacket.
—Letting the door of the disk drive snap shut. This may pinch the diskette. Close the door gently without forcing it.
—Bending or crimping diskettes.

Figure 7-10. Floppy disks with boxes and labels (Courtesy of Verbatim Corp., Sunnyvale, CA)

—Turning the computer off while the disk drive is in use. Wait until the computer finishes using the disk; if necessary, reset the computer if it is stuck in a loop.
—Setting anything on top of the diskettes.
—Dropping anything into the disk mechanism or putting your hands into it.
—Contaminating the diskettes or the drives with dust, coffee, chemicals, soda pop, etc.
—Using diskettes as towels, bookmarks, or Frisbees.
—Using the wrong diskettes for a particular computer. There are many varieties and sizes (see Figure 7-9); be sure to buy the correct ones.
—Expecting the diskettes to last forever. Like other media, they wear out; you should always have copies of important programs and data.
—Letting the diskettes become hot.

Diskettes should be handled like your best phonograph records. With a little care, they will provide much better service than if they are ignored or treated poorly.

KEYBOARD MAINTENANCE

Maintaining a computer keyboard is much like maintaining a typewriter keyboard. Things to do include

—Keep the keyboard clean. Cover it when it is not in use. Clean the keys occasionally with a damp rag.
—Test the key functions regularly to be sure that they are working.
—Mount the keyboard firmly if it is not attached to anything.

Things *not* to do with a keyboard include

—Don't drop anything into or on it.
—Don't contaminate it with coffee, soda pop, crumbs, or other sticky substances.
—Don't beat on it as though it were a manual typewriter. Stroke it as gently as possible; it will last longer and work better.

CRT MAINTENANCE

Maintaining a CRT is like maintaining an ordinary television set. Things to do include.

—Provide proper ventilation.
—Clean the screen regularly with a soft cloth and a nonabrasive cleaner such as liquid glass cleaner.
—Verify proper operation of all functions on a regular basis.

Things to *avoid* with a CRT terminal include:

—Setting the brightness too high.

—Leaving the terminal on for long periods with only the cursor displayed. You may burn the image of the cursor into the screen.

When working on a CRT terminal, provide adequate room lighting and avoid eyestrain, particularly if you have never used a CRT before. A green screen and large, distinct characters help reduce eyestrain.

PRINTER MAINTENANCE

Printers create the most maintenance problems in computer systems. They contain more mechanical parts than do keyboards or video displays and are therefore less reliable. Besides, users always have trouble with paper—adding it, adjusting it, and dealing with jams and misalignment. Ribbons, ink cartridges, and typing elements can also create problems.

A user who depends on a printer should buy the best one he or she can afford. The key factors are high reliability and the availability of local service (usually guaranteed by purchasing a service contract). If you just need occasional program listings or can get along without the printer for a while, a bargain printer may be fine and a service contract makes little sense.

As for maintaining a printer, the things to do are

—Follow the maintenance procedures in the manual. Clean and lubricate the printer as indicated; such service is usually part of a maintenance contract.
—Use high-quality ribbons and paper.
—Insert paper carefully. Be sure that it is aligned properly, both at the sides and at the top and bottom.
—Ensure proper ventilation, particularly if you isolate the noise from the printer with a cover or shroud.
—Test all options regularly.
—Keep the output from piling up or blocking the flow of paper from the printer. Baskets and other holders are often available.

Things *not* to do are the following.

—Don't force anything. Don't try to force the paper in or out or do anything to the mechanism.
—Don't expect an inexpensive, light-duty printer to operate continuously and produce massive amounts of output.
—Don't drop anything into the mechanism.
—Don't put your hands into the printer while it is operating.
—Don't turn the printer off while it is printing.

High-quality printers often have self-testing features that can be helpful. You can also eliminate some operating problems by acquiring a printer stand and a tractor feed mechanism which holds the paper.

RECOGNIZING PROBLEMS

Sooner or later, something will go wrong with your computer. For example, dirt or other contamination, problems in the power lines, or human error may

make the machine misbehave. If this happens, you should first try to resume or restart what you were doing. If that proves impossible, you should reset the computer and start all over again. You can anticipate minor problems by saving programs, data, and other files regularly to simplify recovery, copying and labeling disks and cassettes, marking program listings, and documenting programs (see Chapter 4). Problems will occur occasionally, but they will not have serious consequences if you work carefully and keep adequate records. A computer differs from a tape recorder or television set in that you may have a great deal of work in a vulnerable state. You should try to minimize the amount you could lose. Of course, if minor problems occur frequently, there is probably something seriously wrong and you should investigate the situation further.

The standard way to check out the computer is to run special test programs. These programs should be short and readily available; each one should check a single basic operation. Some manufacturers supply such test programs (often called *diagnostics*); if no diagnostics come with your computer, you should purchase some or write your own. Typical diagnostics are memory test programs that change each word of memory and see if it can be read back correctly, CRT or printer output programs that display or print all the characters, and cassette or disk test programs that record some data and see if it can be read back correctly.

ISOLATING THE PROBLEM

Even if you do not plan to repair the computer, you should isolate the problem as much as possible. In the first place, this may make the cause obvious. If not, it might help you describe the problem to a repair person over the telephone. At least it might help a service person locate the problem, and that could save you time and money.

The first effort is to determine when the problem occurs. Find out which programs are running and which peripherals are being used. Look for key factors such as length of operation, time of day, previous problems, or other warning signs. Mark all the important information in a maintenance log; this may help you recognize the same problem if it recurs. Check all connections to see that they are firmly in place; a loose cable or faulty connection is a common source of problems. Try to determine which part of the computer is at fault. First, eliminate from consideration all components that you know operate properly. Then start with a minimum operational unit and reconnect other devices until the fault reappears. Test programs and self-testing features can help identify a faulty unit. The more information you can provide about what is wrong with your computer, the more easily you or someone else can repair it or recognize the same problem later.

The easiest way to determine what is wrong with a computer is to compare it with an identical system that works. The availability of identical systems for comparison is a good reason to purchase a computer that is popular locally. It is also a good reason to settle on a single model if you have more than one computer (as is likely in a school or business).

CORRECTING THE PROBLEM—SHOULD YOU TRY TO FIX IT?

Once you have determined what the problem is, the next question is whether you should try to fix it. On the one hand, everyone is familiar with the jokes about people who try to fix things but only make the problems larger and more serious. On the other hand, service for personal computers is often expensive or unavailable. You may have to literally pack the system up and send it back to the manufacturer; the delay could be weeks or months and the cost could be very high.

In general, the advantages of do-it-yourself with computers are similar to the advantages with plumbing, television sets, or automobiles. Do-it-yourself is cheaper. Service calls involve minimum rates as well as labor charges. Shipping a computer is expensive and inconvenient. Also, do-it-yourself can save time, particularly if you are located far from any service facilities. This is critical if you depend on the computer and being without it is expensive. Besides, you increase your knowledge of the computer by working on it; the more you do, the more you will know when later problems occur. Furthermore, repair facilities may be limited or nonexistent.

The disadvantages of working on your own computer are, first, that you may go wrong and damage it. Besides, the work may void your warranty or insurance. Second, you may not be able to reassemble the computer. This is a common situation in cartoons. Finally, you may lack the proper tools and parts. The required tools are expensive and you probably will not use them often enough to make them worthwhile unless you have other needs for them.

We should emphasize again the importance of identifying the problem even if you aren't going to repair it. Identifying the problem will often save time and money.

SERVICE CONTRACTS

Repair service is an important consideration if you depend on a computer. If you plan to use the computer to keep business records or perform other essential functions, you want it to operate as close to continuously as possible. A breakdown or a long service delay could be expensive; the cost of a maintenance contract is good insurance against the losses that could result from a system failure. In many situations, the critical factor is how long the computer is unavailable; the cost of the repair is not nearly as important. Consider, for example, the cost to a business of being unable to send invoices, determine the current status of customer accounts, or produce mailing labels for bills or advertising circulars.

If you lack skills and tools and depend on the computer operating properly, you should only buy a system that has local service with a known record. Of course you should expect to pay a higher price for a more maintainable computer. You should also expect to pay for the availability of service, even if you never use it. A reliable dealer will guarantee service or repair within a specified amount of time. A loaner system or a replacement should be made available if the deadline cannot be met.

Many applications do not require high reliability. A computer that is used for learning, teaching, or game playing can be out of commission for a while

without serious loss. You would no more need a maintenance contract for such a computer than you would need one for a television set or a refrigerator.

RESTORING OPERATIONS

Regardless of who has fixed the computer, be sure to verify that it works properly. This is particularly important if a service person has repaired the machine; you should determine whether the computer works before the service person leaves. The way to do this is to have a short test program available. The diagnostic programs mentioned previously can be useful. Having a working system for comparison always helps.

WHAT SHOULD YOU KNOW TO ATTEMPT REPAIR?

If you intend to fix a computer, you should be familiar with the following items:

The computer manuals (Figure 7-8), particularly the schematics and any troubleshooting hints that are provided. Most computers come with a maintenance manual. If you cannot understand that manual, you should call a repair service.

Basic electronics. Courses are available through university extensions, community colleges, trade schools, and adult education centers. Home study courses and books are widely advertised.

Reading schematics. This will be a part of basic electronics courses.

Digital electronics. This involves the use of integrated circuits and requires basic electronics. Courses in this subject are widely available.

Using instruments such as a voltmeter, logic probe, and oscilloscope. Courses in this are also widely available and home study courses and books are widely advertised.

If you are not familiar with the topics listed above, learn them before you attempt computer repair. There is no sense going into the fight unarmed. Popular textbooks and learning aids include military and commercial training manuals and the home study courses offered by Bell and Howell, Cleveland Institute of Electronics, Heath, and NRI (National Radio Institute).

THE REPAIR ENVIRONMENT

If you are going to repair a computer, you must have an adequate workplace. The top of a small desk is not good enough. Features that you should watch for are adequate local and overhead lighting; room for documents, manuals, and schematics; an uncarpeted space with lots of electrical outlets (heavy-duty power strips are helpful); and storage for tools and parts. You should have enough room so you do not have things piled on top of one another.

The minor tools that you will need include the following, all of which are available at a hardware store, an electronics supply store or a computer store: pliers, side cutters, wire strippers, screwdrivers, nut drivers, sharp knife,

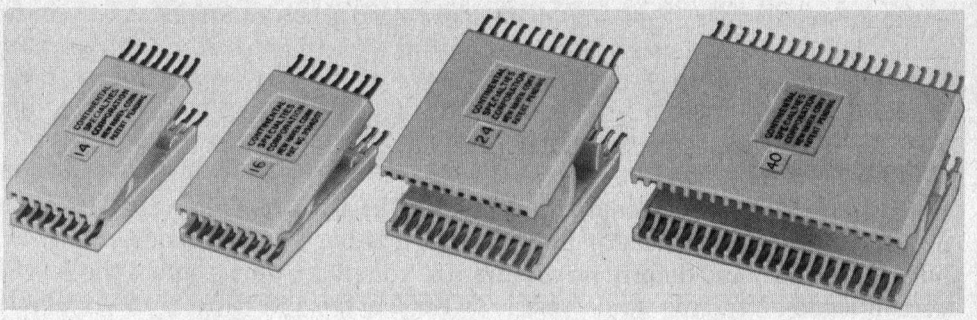


Figure 7-11. Global Specialties' Proto-clip integrated circuit test clips (Courtesy of Global Specialties Corp., New Haven, CT)

soldering iron of the proper size, desoldering aids such as "solder wick," test clips (Figure 7-11), and test cables.

If you are not familiar with soldering, practice for a while before you do it for real. Soldering requires skill and bad soldering is the most common problem with equipment that has been constructed from kits.

You will also need a reasonable set of instruments. Hobby-grade versions are available from Heathkit (Figure 7-12), Global Specialties, Radio Shack, and other suppliers. You will need a *voltmeter* (actually a VOM or volt-ohm-milliammeter) that can be used to measure voltage, resistance, and current (see Figure 7-13); a *logic probe* (Figure 7-14) that allows you to determine whether there is a pulse at a particular point in a circuit as well as its logic level (0 or 1); and an *oscilloscope* (Figure 7-15), preferably a dual-trace one with a bandwidth of at least 5 MHz for use with most computers.

Figure 7-12. The Heathkit Summer Catalog for 1979. Heath is a leading supplier of hobby-level electronic components and test equipment. (Courtesy of Heath Co., Benton Harbor, MI)

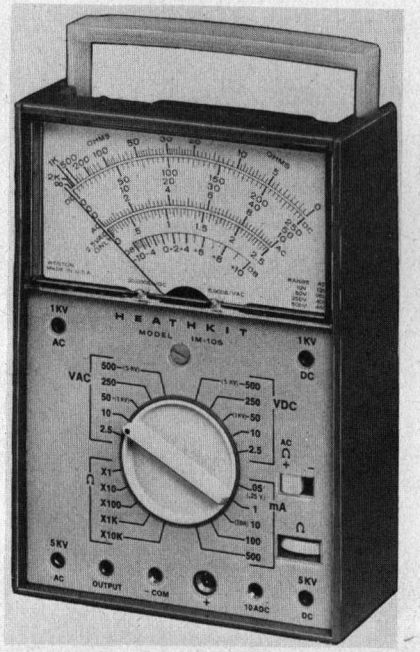


Figure 7-13. The Heathkit IM-105 general-purpose VOM (volt-ohm-milliammeter) (Courtesy of Heath Co., Benton Harbor, MI)

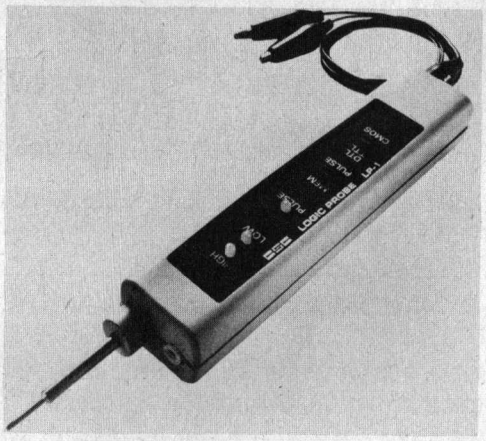


Figure 7-14. The Global Specialties LP-1 logic probe (Courtesy of Global Specialties Corp., New Haven, CT)

An extension of the oscilloscope for complex circuits is the *logic* or *bus analyzer*. The most popular units are the ones from Paratronics (Figures 7-16 and 7-17); they allow you to examine eight lines at once, while an oscilloscope only allows you to examine one or two at a time. Another useful addition, if you have a modular computer, is an *extender board* (in the rear of Figure 7-18) that allows you to bring the faulty board to a position where you can reach it.

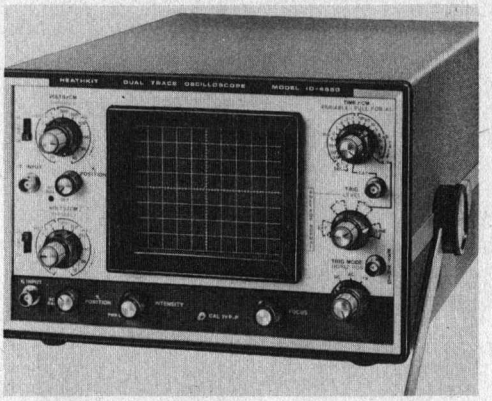


Figure 7-15. The Heathkit IO-4550 dual-trace 10MHz oscilloscope (Courtesy of Heath Co., Benton Harbor, MI)

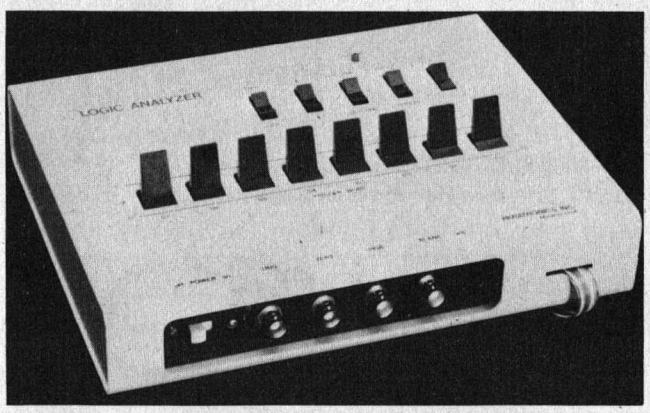


Figure 7-16. The Paratronics Model 100A logic analyzer (Courtesy of Paratronics, Inc., San Jose, CA)

SOFTWARE MAINTENANCE

Clearly, maintaining a computer is like maintaining a typewriter or stereo system. But software maintenance is a different matter. In the first place, the whole idea sounds strange—surely a program doesn't wear out or get out of alignment. The problem is that all programs, whether you buy them or write them yourself, contain errors. Things happen that even the most experienced programmer could not foresee. Furthermore, users will suggest improvements, new peripherals or software will require changes, and programmers will find new ways to do things. The resulting revisions make it necessary to identify most programs by a version number as well as a name; you must be careful that you have the proper version of a popular program (such as the CP/M operating system) since the differences between versions may affect the operation of the computer.

Thus software maintenance consists of identifying errors in programs and suggesting and implementing improvements. Some programs even come with

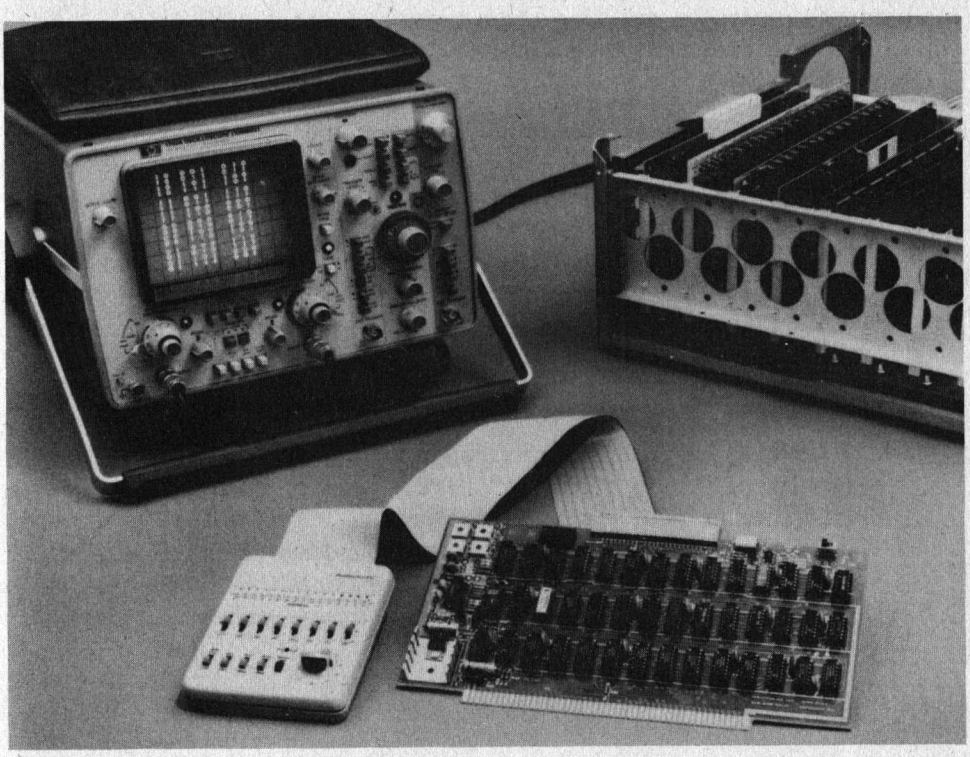


Figure 7-17. The Paratronics Model 150 bus grabber for S-100 bus computers (Courtesy of Paratronics, Inc., San Jose, CA)

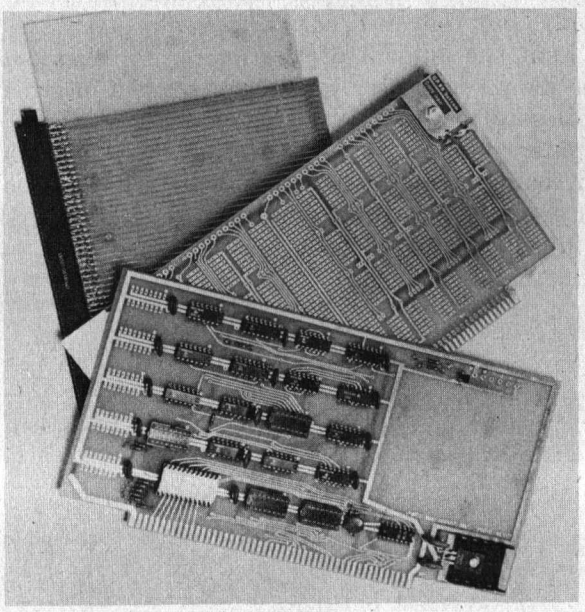


Figure 7-18. An extender board (in the rear) together with a priority interrupt board and a clock board (Courtesy of IMS Manufacturing Co., San Leandro, CA)

maintenance contracts, entitling the user to assistance with problems or to copies of corrections and new versions.

Here are some rules to follow in maintaining software.

—Give each revision of an important program a version number. Mark the number on all copies of the current version and destroy all old versions unless you absolutely must keep them. Few things are more annoying than having an old error come back to haunt you.

—Organize the documentation so you can find it and understand it. If the program is yours, be sure that the documentation meets recognized standards. If the program is purchased, annotate the documentation as you use the program.

—Discover as much as you can about any problems that occur. Identifying the exact conditions may be helpful in locating the error or in describing the problem to the vendor.

—Notify the vendor if you find a truly puzzling problem. Unless your computer or your application is unique, others will have the same problem and the notification will help you, other users, and the vendor.

—Test new programs and corrections immediately. An early trial means that all information is still current and readily available.

SUMMARY

This chapter has described the proper care of your computer. If you take sensible precautions, your computer will operate efficiently and reliably. Most of the procedures are similar to those you would follow in maintaining a household appliance, stereo system, or automobile. Momentary problems will occur occasionally, but they will not cause major setbacks if you are well-organized and limit the amount of data or output that could be lost. If a more serious problem occurs, try to gather as much information as you can about it, regardless of whether you plan to fix it yourself. If you want to repair the computer yourself, be sure that you have the proper background, workplace, and tools. Do-it-yourself can be advantageous and enjoyable if you approach it properly; it can be destructive and a waste of time and money if you leap into it unprepared.

8 Selecting a Computer

The number of separate kinds of personal computers and accessories is already overwhelming. More than thirty computer manufacturers advertise regularly in the popular magazines. If we consider peripherals, software, and other accessories, we are talking about hundreds of suppliers and thousands of products. As is usual in a new business, there are few standards or companies with well-known names and good reputations. Adding to the confusion is technical jargon that often sounds like an obscure foreign language to a newcomer.

We cannot, of course, provide a complete consumer's guide to computers. Such a guide is necessary, but it would have to be updated continuously to cope with the rapid introduction (and disappearance) of products. Instead we discuss the key questions that you should ask in selecting a computer, the places where you should look for equipment and information, and the ways in which you can select a computer for your task. We also present three case studies that illustrate the choice of a computer.

THE KEY QUESTIONS

The key questions to ask in selecting a computer are

—What do you plan to do with the computer?
—How much do you want to spend?
—How much computer background do you have?
—What must you buy and what can you do for yourself?

The important point is to choose a computer that satisfies your needs and that you enjoy using. Beware of well-intentioned advice from people whose talents, backgrounds, pocketbooks, and requirements differ from your own. Unique factors are often important: a store that's right around the corner, a friend or neighbor who already has a particular computer, a special deal, or a possible application in your job may outweigh all the comparisons in books or magazines.

BUYING FOR YOUR APPLICATION

Most computers, like most automobiles, are best suited to particular tasks. You do not want to buy a subcompact to haul a large trailer or a full-sized

station wagon to drive yourself to work and back.

Some applications that might require different computers are:

—Business recordkeeping
—Game playing
—Computer-aided instruction
—Word processing
—Music synthesis
—Controlling factory equipment or machines
—Learning how to program a computer
—Learning how to design or repair products that contain computers
—Learning how to repair computers
—Laboratory automation
—Financial planning and simulation
—Graphics and computer art
—Personal recordkeeping
—Business calculations
—Scientific and engineering calculations

The important question is, what makes these applications different? Here are some obvious factors.

- How much memory is needed? Computers intended primarily for learning or as controllers need far less memory than computers intended for business recordkeeping, word processing, or financial planning.
- What peripherals are required? A computer that is to store large amounts of data needs a mass storage device like a cassette recorder or a disk (see Figure 8-1). A word processing system or a billing system needs a high-quality printer. A computer used for game playing should have a video display. Computers used for instruction or laboratory automation do not require

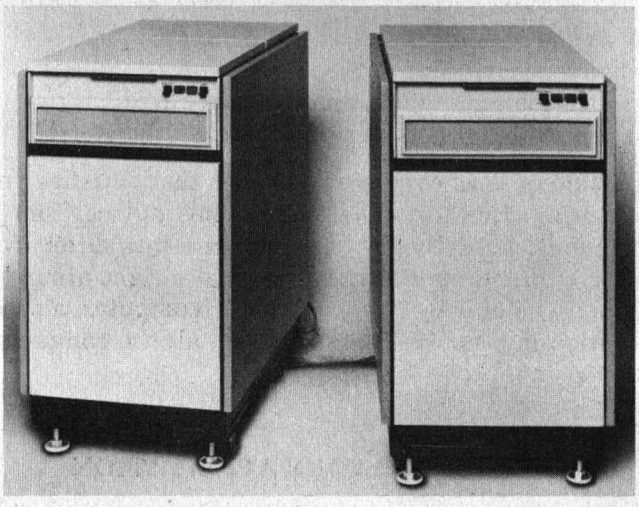


Figure 8-1. The Alpha Micro AM-400 Hard Disk Subsystem (Courtesy of Alpha Micro, Inc., Irvine, CA)

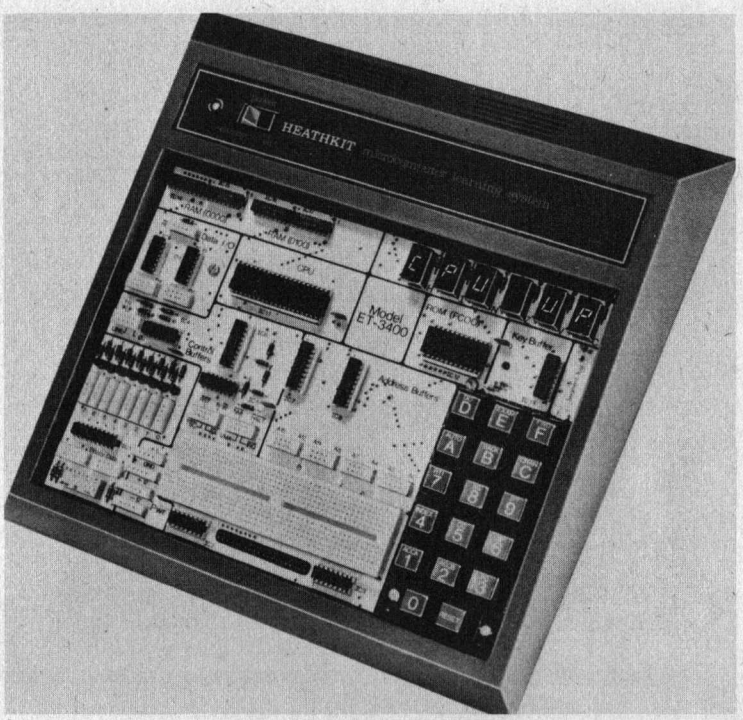


Figure 8-2. The Heath ET-3400 Microcomputer Learning System (Courtesy of Heath Company, Benton Harbor, MI)

high-speed or high-quality peripherals (see Figure 8-2).

- Do things have to be attached to the computer? A computer that is to be used as a controller must have a plug to which you can attach whatever is to be controlled (see Figure 8-3). Computers used in games, computer-aided instruction, word processing, or accounting can be self-contained, like the portable system in Figure 8-4. All you need then are standard peripherals.
- How many programs are required and how much memory do they need? Learning and control applications require fewer and shorter programs than do accounting, word processing, and financial planning. With games and computer-aided instruction, you can start with just one or two programs and then write or purchase others. An accounting system or word processing system, on the other hand, requires a large number of complex programs to be useful. Both number and size of programs are important; together they determine the cost of the programs, the amount of memory required, and the amount of time spent loading programs.
- How reliable must the computer be? As we mentioned in Chapter 7, a computer that you depend on for critical applications must be highly reliable. You will want a maintenance contract if a breakdown could be expensive. Reliability is not nearly so important in applications like learning and game playing.
- What language will be used to write programs? Most controllers and industrial applications are programmed in assembly language. Assembly language, however, is impractical for game playing, word processing, business

Figure 8-3. The Fairchild F-8 Evaluation Kit (Courtesy of Fairchild Micro Systems, Sunnyvale, CA)

data processing, or computer-aided instruction. Here a high-level language such as BASIC or PASCAL is essential.

- What constitutes a reasonable starting system? As we mentioned, word processing and business data processing require a more substantial initial system than do computer-aided instruction, learning about computers or programming, or laboratory automation. You must define exactly what you need. Note, for example, how many components the working system in Figure 8-5 requires.
- How much access to the electronics is necessary? If you want to use computers as controllers or if you want to learn about repair, you must have ready access to the electronics. On the other hand, you would not want ready access in a classroom or business application. A bare circuit board like the one in Figure 8-3 is ideal for the electronics hobbyist but of little value to the mathematics teacher or business person.
- How important is future expansion? A computer intended for classroom use, learning, or control applications may never be expanded. A business system, on the other hand, must be able to grow as the business expands or as the number of applications increases.

Your initial investment is an important factor here. You would not want to outgrow quickly computers that cost several thousand dollars, like those in Figures 8-4 and 8-5. On the other hand, you should not expect to expand simple computers like those in Figures 8-2 and 8-3 that cost only a few hundred dollars. Like remodeling a house, expanding a computer is not worthwhile unless it has a solid foundation and you have an overall plan. Otherwise you are likely to end up with a system that costs more than an off-the-shelf computer but doesn't work as well and isn't worth as much.

Figure 8-4. The iCOM Attache Desktop Computer (Courtesy of Pertec Computer Corporation's Microsystems Division, Woodland Hills, CA)

Figure 8-5. A complete Heath Computer system, featuring the H8 Computer, H9 Terminal, H14 Printer, and H-17 Dual Floppy Disk System (Courtesy of Heath Co., Benton Harbor, MI)

ESTIMATING COSTS

Certainly the amount of money that you can spend will affect your choice of a computer. The following steps will help you use your money wisely.

Determine the minimum system that you need for your application. If this is still more than you can spend, you must either wait or try a simpler application. Fortunately, computers are one of the few areas where prices are coming down; you don't have to buy now to beat the next price increase.

Rank extra accessories in the order of their desirability. Then you can add to your initial system in a reasonable way.

Estimate how much you would be willing to pay for the various accessories. Some may be far more expensive than you expect, whereas others may cost less. You need an estimate of what the accessories are worth to you.

When you shop for a computer, remember all the things that a useful system must have. Don't be impressed by a low sticker price if the product lacks some essential components. A computer requires

—Hardware
—Systems software
—Applications software
—Peripherals
—Memory
—Working environment
—Maintenance
—Service
—Expendable supplies
—Incidentals such as cords, cables, storage bins, and connectors

Don't ignore the cost of any of these items. Make some reasonable estimates before you go shopping. You can usually find prices in magazine advertisements, but remember that advertisers generally emphasize their lowest conceivable prices and may not include some essential items. So add an extra percentage as a margin. Don't forget to include postage, handling, shipping, and taxes.

BUYING FOR YOUR BACKGROUND

An important consideration in buying a computer is your background. If you are a newcomer to computers, you should not buy a system that you cannot use. You may want to start by purchasing a computer that will do only a simple job or that serves only as an educational tool. Remember, computers scare many people; an inexpensive computer that helps overcome those fears can easily be worth its cost. Later you will be ready to select and use a more powerful system.

To determine whether a computer is suited to your background, ask yourself the following questions.

- Can you read the manuals? Don't expect to understand everything in them, but step down a level if they sound as if they're written in a foreign language. Many manufacturers sell manuals separately at nominal cost and may even refund that cost if you buy their products later.
- Do you know what the peripherals do and how they operate? Simple peripherals like keyboards, cassette recorders, and television sets are more familiar to the newcomer than are floppy disks, printers, and graphics displays.
- Can you start and stop the system? If starting the computer is complex and mysterious, the system is surely more than you want.
- Can you load programs and data? If you need help every time you want to load a new program or enter some data, you are in over your head.

You need not be an expert on your computer. The system should be capable

of much more than you can utilize initially. It should have many features for you to investigate. But you must be able to get started: if you can't read the manuals or get the computer to perform the simplest operations, you are asking for trouble. Don't be ashamed of your ignorance. People have survived and prospered (amazing!) without knowing a thing about computers. However, buying a computer that you don't understand is risky and often leads to discouragement and frustration.

DECIDING ON YOUR OWN CONTRIBUTION

Besides maintenance and service (which we discussed in Chapter 7), other tasks that you may want to perform by yourself include writing programs (remember Chapter 4), constructing hardware from a kit like the one in Figure 8-6, and building your own peripherals or interfaces. Of course, the do-it-yourself approach may or may not be right for you. On the bright side, you can save time and money, improve your own skills, make the computer do exactly what you want, and provide yourself with enjoyment and satisfaction. But do-it-yourself requires skill, time, and money; it seldom results in a professional-quality product; and it can be frustrating, exhausting, and even dangerous.

One problem with computers is rapid technological change. Your skills and your homemade equipment may soon be obsolete. Furthermore, you may

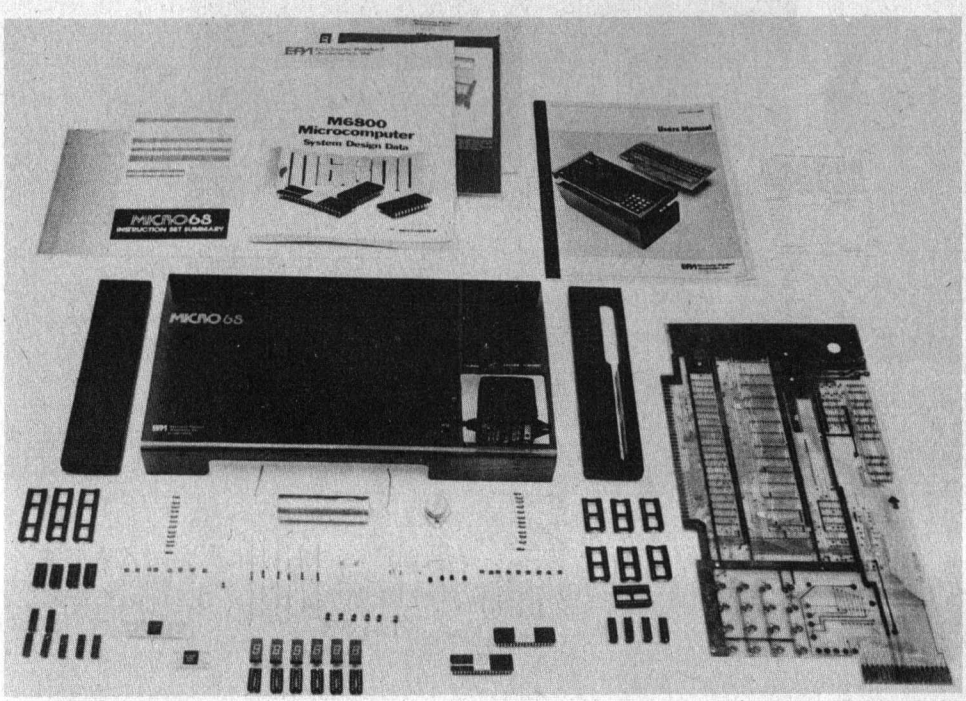


Figure 8-6. Components in an EPA Micro-68 computer kit (Courtesy of Electronic Product Associates, San Diego, CA)

Figure 8-7. The Heathkit Model IM-5238 AC Voltmeter (Courtesy of Heath Co., Benton Harbor, MI)

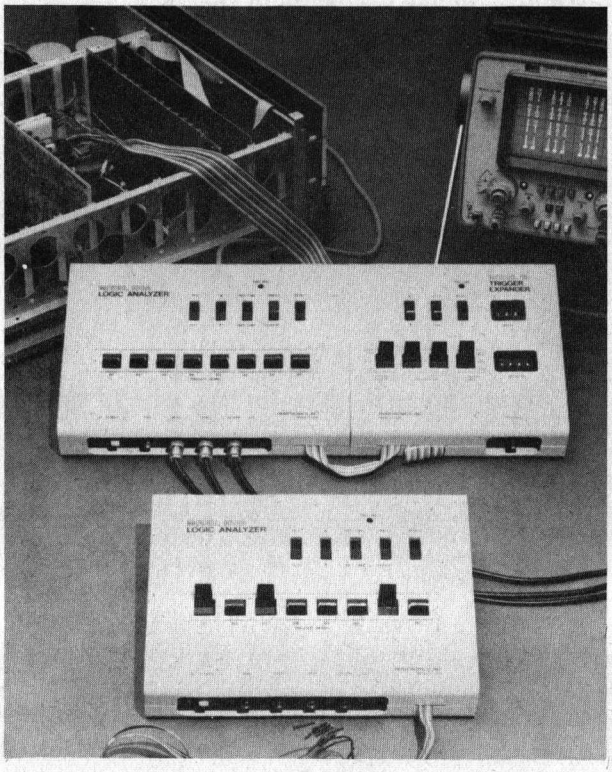


Figure 8-8. The Paratronics Model 100A Logic Analyzer (Courtesy of Paratronics, Inc., San Jose, CA)

find that what you built is now available for less than it cost you to make it. This problem has plagued computer kitmakers. By the time they have designed, documented, packaged, and distributed a kit, you can buy a better finished product cheaper than you can buy the kit. So be careful that your projects aren't just a waste of time.

Before you start a project, ask yourself the following questions.

—How much does the finished product cost?
—What tools will you need? Can you use them on other projects? Figures 8-7 and 8-8 show typical tools.
—How long will it take you to complete the project? If you haven't any idea, you may be better off not starting.
—What will you learn by doing this project? If you learn something important, you will have profited even if you never use the product or if you find that you could have bought it for less than it cost you to make it.
—Can you check out the project? If you have no way to test your work, you will never be sure of what you can do or be able to rely on the completed product.
—How much time can you devote to the project? Even an inexpensive kit is no bargain if it sits half-completed in your garage.
—How much help can you get? Friends who have completed the same project, some examples to look at, or some help from a club or a store can mean the difference between success and frustration.

We don't want to discourage you from trying to do things on your own. But you should choose projects that you have a chance to complete. Don't overestimate your skills or your free time. Be particularly careful of hardware projects that may require a large amount of electronic test equipment; working on such projects without adequate equipment is frustrating and a waste of time.

PLANNING FOR THE LONG TERM

You must consider your long-term plans when you buy a computer. The following questions can help you determine the long-term value of a computer.

• Is it a popular model? If it is, you are likely to find information, service, other users, and suppliers of accessories and software. If it is not, you will be on your own. Look at the list of dealers and check the computer magazines for other suppliers. Computer clubs often poll their members to find which systems are most popular locally. Table 8-1 contains the results of a survey of the Amateur Computer Group of New Jersey (courtesy of Sol Libes).

• Does the manufacturer (or other supplier) provide for expansion? Are more peripherals, extra memory, and more software readily available?

• Does the computer have a standard language and a standard interface? These greatly simplify expansion. Figure 8-9 shows a popular computer that uses the S-100 bus and can be programmed in BASIC or FORTRAN.

• Does the manufacturer plan compatibility with future products? Certainly the technology will change and you won't want to be left behind. Some

Table 8-1. 1979 Survey of Members of the Amateur Computer Group of New Jersey (Courtesy of Sol Libes)

Members' Occupations			
Electrical Engineer	173	Businessman	15
Electronics Technician	76	Retired	14
System Analyst	60	Doctor/Dentist	11
Programmer	60	Attorney	9
Student	42	Writer/Editor	7
Teacher	37	Electronic Sales	7
Chemist/Engineer	27	Physicist	7
Data Processing Manager	25	Marketing Manager/Analyst	7
Consultant	18	Publisher	5
Business Manager	17	Other	61
CPUs			
1802	35	Interdata	5
9900	11	SC/MP	5
8085	8	LSI-11	5
Nova	4	Other	14
Computers			
8080		**Z-80**	
Imsai	101	Radio Shack TRS-80	97
Processor Technology (SOL)	31	Technical Design Labs (TDL)[1]	67
Altair	23	Cromemco	27
Homebrew	13	Ithaca Audio	25
Heath	10	Exidy Sorcerer	8
Polymorphic Systems	8	Digital Group	7
Others	16	SD Sales	6
		Others	13
6800		**6502**	
Southwest Technical Products	38	MOS Technology KIM	61
Motorola Evaluation Kits	20	Commodore PET	36
Homebrew	7	OSI	21
Heath	6	Apple	21
Sphere	4	Synertek SYM	4
Altair	2	Others	8
OSI	2		
Others	4		
Memory Size			
RAM		ROM	
64K	2	32K	3
60-64K	25	20-24K	4
50-59K	28	16-18K	14
40-49K	60	12-14K	48
30-39K	69	8-10K	34
20-29K	43	3-6K	50
10-19K	53	1K-2K	92
Less than 10K	50	Less than 1K	24

Peripherals

Hardcopy Terminals

TTY (33)	94	DECwriter	23
TTY (40-43)	26	Flexowriter	10
TTY, other	38	Portacom	5
Selectric	36	Qume	4
Diablo	32	GE Terminet	3
Texas Instruments	24	Other	8

CRT Terminals		**Printers**	
Lear Siegler ADM-3	34	Centronics	27
Hazeltine	17	Southwest Technical Products	17
Heath	13	Integral Data	5
Homebrew	11	Okidata	3
ASCIIscope	11	Expandor	3
ACT-I/IV	6	Others	23
Others	31		

Video Display Modules/Television Terminals		**Graphics**	
VDM-1	91	Southwest Technical Products	11
Southwest Technical Products	60	Homebrew	4
VT-1	18	Others	9
VDB	17		
VB-1	12		
Xitex	5		
Others	30		

Cassette		**Disk**	
Tarbell	71	Tarbell	66
CUTS (Processor Technology)	36	North Star	55
Technical Design Labs (TDL)[1]	26	Radio Shack TRS-80	33
Southwest Technical Products	25	Micropolis	19
Homebrew	19	iCOM (Pertec)	16
National Multiplex	17	Southwest Technical Products	11
MITS	8	Thinker Toys	8
Imsai	7	Homebrew	5
Polymorphic Systems	6	Cromemco	4
Others	28	Others	40

Other Equipment		**Applications** (first 200 membership forms)	
Modem	90	Games	129
Digital-to-analog converter	47	Software Development	109
Analog-to-digital converter	43	Business	72
Voice input	15	Education	65
Voice output	11	Word processing	59
Music synthesizer	14	Hardware Development	46
Dazzler	12	Scientific	41

Other Equipment		**Applications** (first 200 membership forms)	
Real-time clock	5	Graphics	34
Plotter	3	Music	31
		Process Control	27
		Ham Radio	25
		Robotics	5

[1]Technical Design Laboratories (TDL) was a local New Jersey manufacturer that is no longer in business.

manufacturers have a proven record of upgrading their older systems at reasonable cost.

- Is the manufacturer a sound company that is likely to survive and expand? Many manufacturers are small but you can still judge whether their products are well-designed and attractive and their organization is adequate.
- Are the manufacturer and dealer honest and cooperative? If you have problems at every step, you can expect more trouble in the future. Mistakes and delays may be excusable but a basic lack of cooperation and honesty is not.
- Is the system locally popular? There are geographical variations, and particular computers may have strong local dealers, user's groups, or proponents.

Your intended use is critical in determining which considerations are most important. You do not want to pay extra for a computer that follows standards and provides for expansion if you do not intend to use the standards or expand the computer. But if you plan to keep your computer and depend on it heavily, be sure that it is in the mainstream and that the manufacturer has adequate plans for keeping it there. If you have developed a system like the one in Figure 8-10, you do not want it to become quickly obsolete or unserviceable.

How do you cope with rapid technological change? Won't new products be available soon that will make the current ones obsolete? The answer is that new products will be available and current computers will become obsolete.

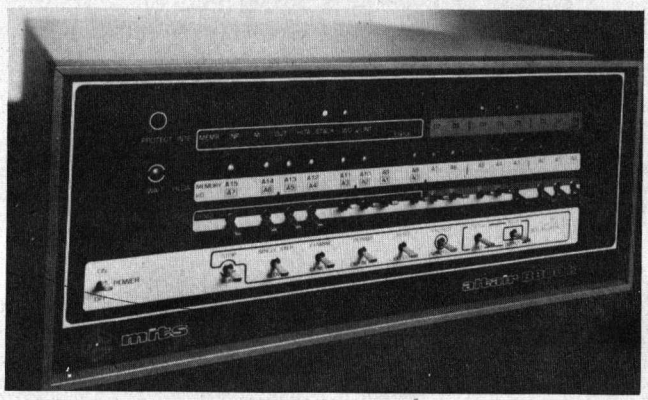


Figure 8-9. The Altair 8800b Microcomputer (Courtesy of Pertec Computer Corp., Microsystems Division, Albuquerque, NM)

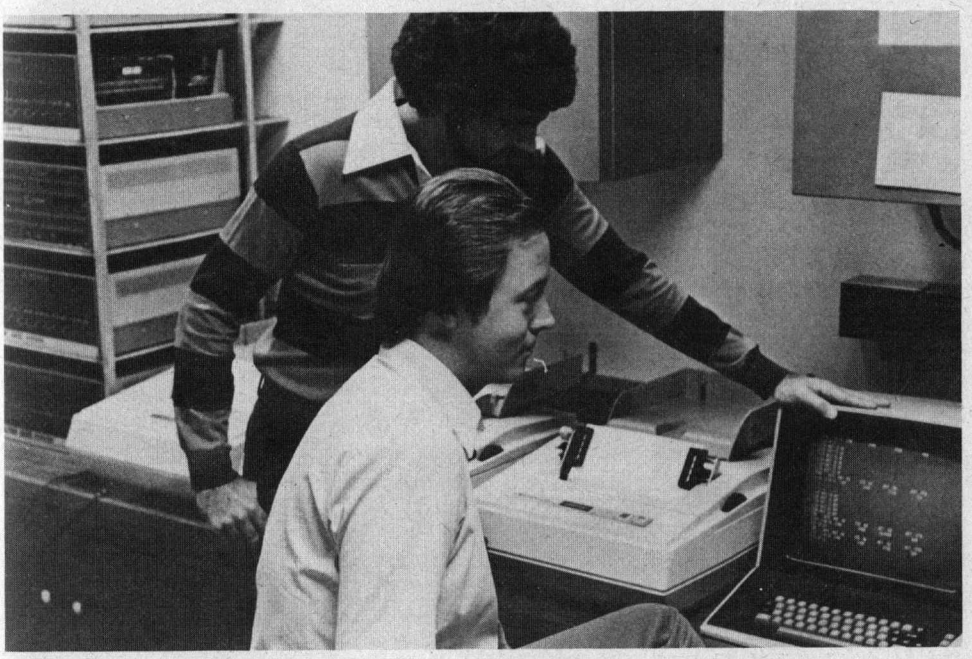


Figure 8-10. Two members of Knott's Berry Farm's Construction and Maintenance Division check operations using one of their Altair microcomputers (Courtesy of Pertec Computer Corp., Los Angeles, CA)

In general you can expect a new generation of computers every ten years or so. This means that you should beware of systems that have been on the market more than five years; they could be outmoded unless the manufacturer has upgraded them.

You should not, however, underestimate the value of just learning about computers. You can read about them, of course, but there are many things you can learn only by owning one and using it. Many people find computers mysterious and frightening; for them, learning how to operate and use one can be worth a great deal. The fact that your computer does not have the latest gadgets or could be bought now for less than you paid for it may not matter. If you learn the ins and outs of hardware, software, operating systems, compilers, editors, debuggers, data bases, and peripherals, you will be ready to purchase a new computer intelligently and use it effectively.

Computer technology is not slowing down. Ahead of us could easily be another fifty years of spectacular progress. Waiting for the best computer at the lowest price could mean waiting for the rest of your life. You might as well jump on the bandwagon now.

The real question is whether you can do what you want with today's equipment at a price you can afford. If you can, the computer will be worth its cost and you should not worry about what you might have bought or how much less you might have paid for it. Remember, computers will not go away. If you don't buy one now and learn how to use it, you will simply be faced with the same choices later. Meanwhile others (your friends, coworkers, competitors) will have moved ahead of you. The most important long-term consideration is that computers will become commonplace; those who learn

about them early will be ahead of the game, while those who wait too long will find themselves, not their computers, obsolete.

Computers are fascinating. If you buy one that is well-suited to your background and intended use, you will be far too busy and having far too much fun to worry about the changing technology.

OTHER QUESTIONS

There are many other considerations in buying a computer, such as the following questions.

- Is the dealer (or manufacturer) local? It's easier to get things right if you can drive over rather than having to make a transcontinental telephone call.
- Do you know people who own the same machine? Not only can you share experiences and problems, but you will also have a working system for comparison if yours malfunctions.
- Are there books and magazine articles about the system? These can help you learn about your computer and how to use it.
- Has the computer been used in applications like yours? If so, you will be able to talk to other users and get support from the manufacturer. If not, you may find yourself breaking new ground all the time.
- Are there courses or study guides for the computer? They can make learning much easier.
- What warranty is offered? How do you obtain warranty service?
- How can the system be maintained? Is a service contract available?
- What other support does the manufacturer or other suppliers offer, such as a user's group or newsletter? Is there a local user's group?
- Can you obtain a system off-the-shelf? If not, when will the system be delivered?
- Are there additional suppliers of compatible accessories and software? This may have a major effect on their cost and availability.

Note some of the tactics that you must be wary of.

—An incredibly low price that does not include essential hardware or software
—A promise of providing hardware, software, or a packaged system without a guaranteed completion date and penalties
—Warranty service that requires you to ship the system to the manufacturer and pay a fee
—Components that are used, lack documentation, or are manufactured by companies that are no longer in business
—Untested software or hardware
—A combination of components from many different manufacturers and suppliers without a guarantee of compatibility
—Peripherals and software that depend on other hardware or software that may cost extra or may not even be available
—A system that is difficult to change, expand, or maintain
—Peripherals that lack interfaces or software

Many of the questions we have mentioned are important if you are an inexperienced user working on a common application, such as maintaining inventory records or automating a mailing list. If you have computer experience, your main concern should be how much capability you can get for your money. The manufacturer's support is unlikely to be useful. If your application is unique, your main concern should be whether the computer is suited to it. The manufacturer is unlikely to provide more than friendly advice. You should not look for support when you don't need it or are unlikely to find it.

SOURCES OF INFORMATION

The following lists suggest sources where you may find more information; we have grouped them according to how likely they are to satisfy your needs.

Good sources

—The local computer club
—Local computer stores
—Computer hobbyist magazines
—Computer shows

Questionable sources (Depends on where you are. This is not a criticism of these sources, but providing information about personal computers is not their primary business and their interest and expertise vary greatly.)

—Electronics hobby stores
—Technical institutes or schools of engineering
—Electronics and popular science magazines
—Local computer manufacturers and distributors
—Professional societies (usually engineering or computer-oriented, but may include business and educational groups as well)

Unlikely sources (Some of these may be good sources in special cases, but you should not count on them. Of course, they may be all that is available in smaller communities.)

—Public libraries (seldom interested or up to date)
—Nontechnical schools (depends on whether faculty members are interested)
—Department stores
—Business equipment suppliers
—Surplus stores
—Adult education and trade schools

COMPUTER MAGAZINES

If you plan to buy a computer, you will surely want to look at one or more of these specialized magazines. Even if you cannot understand the articles or

find nothing of interest in them, you can still read the news and advertisements. The news features will inform you about clubs, computer shows, new products, and events in the industry. The advertisements will tell you about products, prices, local and mail-order suppliers, and special services. But be careful. Even the news and advertisements often contain a lot of jargon.

The leading magazines include those listed here alphabetically with brief descriptions.

1. *Byte:* the first and most widely circulated computer hobbyist magazine intended for experienced computer users. *Byte* also publishes *onComputing* magazine, which is aimed at those who purchase packaged computers rather than build their own.

   Write: Byte
   70 Main St.
   Peterborough, NH 03458

2. *The Computing Teacher:* a magazine intended for elementary and secondary school teachers, particularly those who are just starting to use computers.

   Write: The Computing Teacher
   Oregon Council for Computer Education
   Computing Center
   Eastern Oregon State College
   La Grande, OR 97850

3. *Creative Computing:* a magazine intended for inexperienced users without a technical background. *Creative Computing* also publishes *S-100 Microsystems* magazine, which is aimed at owners of modular computers based on the S-100 bus.

   Write: Creative Computing
   P.O. Box 789-M
   Morristown, NJ 07960

4. *Dr. Dobb's Journal:* a magazine largely oriented toward software for experienced computer users.

   Write: People's Computer Company
   P.O. Box E
   Menlo Park, CA 94025

5. *InfoWorld:* a newspaper for the personal computer industry. Jim Warren, a leader in personal and hobby computing, founded *InfoWorld* and the West Coast Computer Faire.

   Write: InfoWorld
   530 Lytton Ave.
   Palo Alto, CA 94301

6. *Interface Age:* a magazine that specializes in business applications of personal computers, intended for experienced users.

Write: Interface Age
P.O. Box 1234
Cerritos, CA 90701

7. *Microcomputing* (formerly *Kilobaud*): a determined competitor of *Byte* that is also intended for experienced users.

Write: Green Publishing
73 Pine St.
Petersborough, NH 03458

The efforts of Wayne Green have made Peterborough, New Hampshire, a center of both amateur radio and computer hobbyist publications. Green Publishing also produces *TRS-80 Microcomputing,* specifically aimed at owners of the Radio Shack TRS-80 computer.

We may generally characterize these magazines as follows:

Advertising-Rich:
Byte
Interface Age
Microcomputing

Business-Oriented:
Interface Age

Education-Oriented:
Computing Teacher
Creative Computing

Experienced Users:
Byte
Dr. Dobb's Journal
Interface Age
Microcomputing

Inexperienced Users:
Computing Teacher
Creative Computing
onComputing

Newspapers:
InfoWorld

Software:
Dr. Dobb's Journal

There are other magazines devoted to such topics as specific applications—medicine, business, art, music, word processing; specific computers—Radio Shack TRS-80, Commodore PET, Heath H-8, Apple or specific microprocessors such as the 6502 or 6800; and specific functions or services—computer retailing, software, or computer clubs. Still other magazines provide

an informal marketplace, current news, reviews from the hobbyist or technical press, and current lists of manufacturers, programs, or articles.

The most popular magazines are available at computer stores and occasionally at other outlets. Eventually newsstands and libraries should carry them. The leading magazines also advertise heavily, particularly in popular scientific publications and in the engineering and computer trade press.

COMPUTER CLUBS

Although a few small groups have existed since the 1950s, most computer clubs are a product of the late 1970s. They generally publish newsletters and hold occasional meetings. Typical reasons for joining the local club are meeting people with similar interests and equipment; finding out about stores and suppliers; participating in group purchases and learning about equipment and programs from current users; learning about courses, computerized bulletin boards and information exchanges, and other services that may be offered; and obtaining or exchanging programs.

Locating a club is often difficult, since most clubs are small and lack telephone listings. *Byte, Microcomputing,* and *Interface Age* carry club news and even an occasional directory of clubs. Other places to check are computer stores, colleges, and computer manufacturers, distributors, or representatives. The local club is a good place to get started. You are likely to find other people working at a similar level on similar applications.

COMPUTER STORES

The first retail computer store was founded in 1975. Within two years it had grown to a half-million dollar business. Now there are stores in most cities. Some stock only one brand of computer; others handle a variety of products. There are even chains and franchises.

You can find out about local stores from the telephone directory (usually in the yellow pages under "Data Processing Equipment" or "Computer Stores"), local computer club, or a computer magazine. Almost all stores carry computers, peripherals, software, magazines, books, supplies, tools, and parts. Many stores also offer service and classes.

The most widespread stores are franchised operations such as Computerland and Byte Shop. Computerland, for example, offers computers for the hobbyist, consumer, and commercial markets. All their stores provide a service department, standard product lines, and local options such as classes, programming services, additional specialty products, and recommendations for consulting.

The advantages of dealing with a local computer store are

- —Direct contact with people who are knowledgeable about personal computers
- —Local service and information
- —Proven product lines
- —Information about local clubs, suppliers, courses, consultants, etc.
- —Local handling of warranties

—Source for specialty books and magazines
—Source for supplies such as tapes, disks, and paper
—Demonstration systems that you can examine, try out and perhaps even lease or use on a trial basis
—Source of in-stock computers, peripherals, and software

The owners of the local computer store cannot be experts on all types of computers or applications. Nor can they guarantee the quality of all the products they sell or the services they recommend. But a local store provides direct contact and a commitment to continued service. We strongly urge you to visit such a store before you buy a computer. It is a resource you will want to become acquainted with. One of us (LAL) would like to thank Ed Faber and Greg Grumich of Computerland for their cooperation and answers to our questions.

COMPUTER SHOWS

One way to find out about computers and suppliers is to attend a computer show. These are typically held in major metropolitan areas and feature exhibits of personal computer equipment, seminars and short courses, special interest group meetings, films, demonstrations, programming contests, and technical sessions. Often the number of exhibits and events is mind-boggling. If a show is held nearby, this is an excellent opportunity to see many different systems and obtain a large amount of information in one place. Among the leading shows are

West Coast Computer Faire (held in California)

Write: Computer Faire
P.O. Box 1579
Palo Alto, CA 94302

Personal Computing (held on the East Coast)

Write: Personal Computing
Route 1, Box 242
Mays Landing, NJ 08830

Mid-America Personal Computer Show (held in the Midwest)

Write: International Personal Computer Shows, Inc.
P.O. Box 844
Greenwich, CT 06830

National Computer Conference Personal Computing Festival (held in conjunction with the National Computer Conference)

Write: American Federation of Information
Processing Societies, Inc.
1815 North Lynn St., Suite 800
Arlington, VA 22209

Trenton Computer Festival (held in Trenton, NJ)

Write: Trenton Computer Festival
Trenton State College
Trenton, NJ 08625

There are many other shows of varying sizes and orientations. Most of these advertise in the computer magazines.

SCHOOLS

Many schools have been leaders in offering courses on personal computers. The quality and orientation of these courses vary greatly, but we should mention the following schools and active faculty members

—Trenton State College (Trenton, NJ):
Allen Katz
—Union County Technical Institute (Scotch Plains, NJ):
Sol Libes
—DeAnza Community College (Palo Alto, CA):
Leroy Finkel
—Virginia Polytechnic Institute (Blacksburg, VA):
Peter Rony, David Larsen, and Jon Titus

Certainly other colleges, universities, trade schools, technical institutes, and adult education centers will offer courses in personal computers since the equipment is inexpensive and the demand is growing.

MAIL ORDER SUPPLIERS

Many mail-order suppliers offer personal computers and accessories. Such suppliers advertise in the major computer magazines and have displays at the computer shows. As in most industries, some mail-order suppliers have developed bad reputations while others are highly regarded.

We suggest the following approach to mail-order suppliers.

- Ask at the local computer club to see if anyone has dealt with the supplier.
- Try a small order first to test the promptness of the service and the quality of the goods.
- Insist on a definite delivery date with cancellation of the order if that date cannot be met.
- Be wary of suppliers who demand payment far in advance of delivery. Many buyers have lost money when manufacturers and suppliers have gone out of business.

If you are a newcomer to computers, we strongly recommend that you deal with a local computer store. Be wary of extremely low prices offered in advertisements. The merchandise may not be what the beginner would expect; a great deal of work and many additional parts may be necessary to make the items usable. If you are experienced and do not need the support offered by

a local dealer, you may want to take advantage of the lower prices typically charged by mail-order suppliers. But follow the precautions that we mentioned and don't be a sucker.

OTHER SOURCES

Other sources of equipment and information are

- —Radio/electronics stores, such as Radio Shack
- —Electronics hobby suppliers such as Heath
- —Correspondence schools
- —Magazines such as *Popular Electronics, Radio/Electronics, Scientific American,* and *73*
- —Technical and professional groups such as ACM (Association for Computing Machinery), IEEE (Institute for Electrical and Electronic Engineers), AEDS (Association for Educational Data Systems), DPMA (Data Processing Management Association), and NCTM (National Council of Teachers of Mathematics); many of these groups sponsor meetings and conferences that are not highly technical.

FITTING THE COMPUTER TO THE TASK

You want to buy the computer that fits your task. Let us first recall how we divided personal computers into several categories in Chapter 2. We will then discuss the tasks each category can handle and present some brief case studies, emphasizing which factors are the most important in selecting a computer.

We have already categorized personal computers as

- —One-board computers: MOS Technology KIM, Rockwell AIM-65, and Synertek SYM (see Figure 8-11)
- —Appliance computers: Radio Shack TRS-80, Commodore PET, Apple

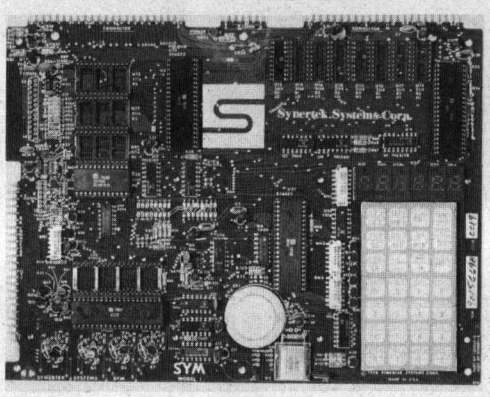


Figure 8-11. The Synertek SYM-1 one-board computer (Courtesy of Synertek Systems Corp., Sunnyvale, CA)

II, Atari 400 and 800, Texas Instruments 99/4, and Compucolor II (see Figure 8-12)

—Fully expandable or modular computers: Altos ACS8000, Cromemco Z-2, North Star Horizon, and Southwest Technical Products SWTP 6800 (see Figures 8-13 and 8-14)

Generally, the various categories of computers are best suited to the following kinds of tasks:

One-board computers are ideal for simple control applications, (e.g., turning the heat on and off or operating a sprinkler system) and for learning how to use microprocessors in industrial products.

Appliance computers are ideal for classroom use, for learning how to program, and for simple, repetitive applications that require a low-cost packaged system.

Modular computers are ideal for situations where the computer is to be used for a variety of tasks, long-term use and significant expansion of the system are planned, and the applications are critical enough to warrant a maintenance contract and a thorough analysis of requirements.

The following are some specific applications for each type of computer:

One-board computers

Workstation for electronics classes
Laboratory instrument controller

Figure 8-12. The Atari 800 Personal Computer System with television, cassette recorder, and printer (Courtesy of Atari, Inc., Sunnyvale, CA)

Figure 8-13. An Altos ACS8000 computer system with dual floppy disks, hard disk, printer, and CRT terminal (Courtesy of Altos Computer Systems, San Jose, CA)

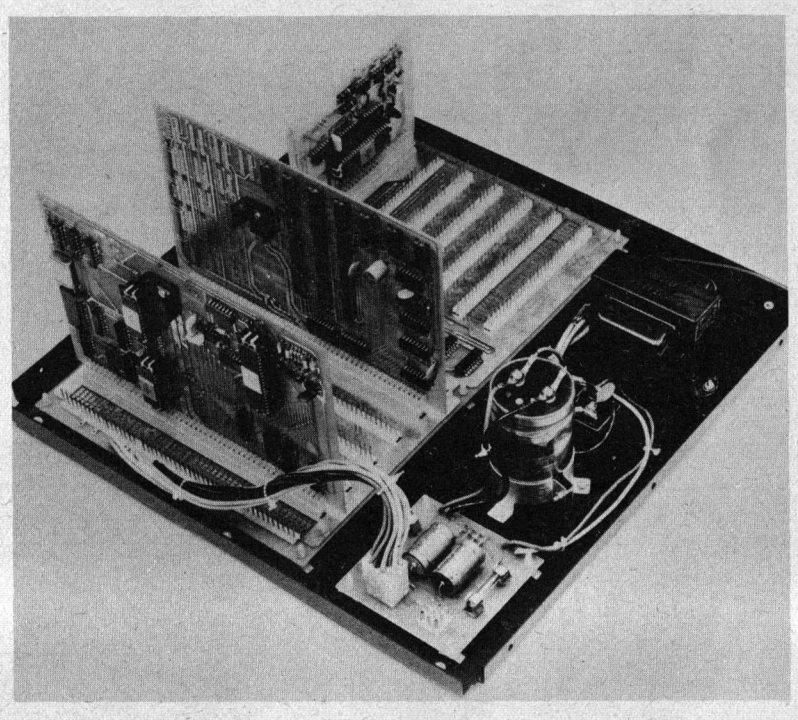


Figure 8-14. An interior view of the SWTP 6800 Microcomputer (Courtesy of Southwest Technical Products Corp., San Antonio, TX)

Prototyping computer for electronics projects
Low-cost remote terminal
Workstation for assembly language programming class
Model train controller
Simple tester

Appliance computers

Drill and practice or game computer for elementary or secondary schools
Workstation for BASIC programming class
Simple word processor
Low-cost terminal for educational use
Programmed video calculator for simple engineering and business tasks
Video game machine
Home computer

Modular computers

Bookkeeping for a small business
Money or investment management
Medical, dental, or legal office management
Time-sharing system (see Figure 8-15)
School recordkeeping
Simulation
Financial planning

Finally, let us look at some typical case studies:

Case One: You are an engineer or technician who will soon be involved in the design of an instrument that uses a microprocessor. You want to buy a computer that will help you learn the new skills required for this project.

Figure 8-15. A time-sharing system that can run up to 8 independent programs at a time (Courtesy of Pertec Computer Corp., Microsystems Division, Woodland Hills, CA)

The criteria for the computer are:

—Low cost, preferably not over $500
—Availability of educational materials aimed at people with backgrounds in electronics (a self-teaching course would be useful.)
—Ready access to input and output for implementing control applications
—Simple operation for small programs
—Relevance to the work project, which involves a specific microprocessor

Here, expansion is unimportant, since you will probably not expand the computer. Nor are peripherals or memory a problem, other than that the computer should have enough to make programming reasonably simple and straightforward.

The solution, of course, is to purchase a one-board computer. Any of these will be inexpensive and easy to operate, and will provide ready access to input and output. The critical factors are relevance to the work project and the availability of educational materials, in that order. The first criterion favors computers based on the specific microprocessor that will be used in the design. Although microprocessors are similar, it makes little sense for someone to learn about one in order to use another. That would be like learning Spanish in preparation for a trip to Italy.

If you still have a choice, the availability of educational materials should help you decide. A quick look at what the manufacturer and publishers offer should enable you to select a particular computer. Materials are usually only available for the most popular one-board computers (such as the MOS Technology KIM, Synertek SYM, and Rockwell AIM), the best supported evaluation kits (such as the Intel SDK-85 and Motorola MEK6800D2), and the computers specifically developed for use in engineering education (such as the Heath ET3400 and the E and L Instruments Mini-Micro Designer). The last group deserves special attention, because those computers are supported by textbooks, workbooks, home-study courses, and other learning aids.

Case Two: You teach at a high school that plans to buy computers for computer literacy and programming classes.

The criteria for the computers are:

—Ease of use, so neither teachers nor students will require a large amount of training just to get started
—Availability of materials for computer-aided-instruction
—Ready access for large numbers of users with limited skills (Each computer must serve a large student population.)
—Availability of complete systems that are ready to use (Few high school teachers want to build computers from kits or even assemble them from component units.)
—Local maintenance and service, so problems can be handled readily without having a highly trained person on the staff.

Here, the key is to provide simple access to a computer for large numbers of students. Expansion and high-quality peripherals and software are not really important, since the aim is education rather than critical applications.

The solution is to buy as many low-cost appliance computers as the budget

will allow. This will provide computer training for a large number of students. The appliance computers are complete, easy to use and explain, locally serviced, and ready-to-go. Note the advantages of many appliance computers over a single large computer:

—The appliance computers are easier to use and explain.
—Each appliance computer is independent, so problems with one do not affect the others and many students can work simultaneously without slowing the response time of a single large computer. Remember also that an appliance computer costs about the same amount as a terminal for a large computer.
—Students can utilize all the computer's facilities. One student cannot interfere with others or damage central components.
—Appliance computers are less intimidating than large computers, since they are aimed at a mass market and their peripherals usually appear simple and familiar.
—Hardware and software for appliance computers is widely available at low cost and is generally easy to use and understand.

While a large central computer is essential for handling administrative tasks or providing access to shared data files, many inexpensive computers are the ideal way to provide basic computer education.

Which appliance computer should you buy? The important issue here is the availability of instructional materials and programs. You can easily determine which computers are the most popular in your community. A look at the software catalogs and discussions with nearby schools will tell you what materials and programs are available.

The magazine *Computing Teacher* describes the experiences of various schools, reviews hardware and software, and discusses educational applications of computers. As we mentioned earlier, this magazine is available from Computing Center, Eastern Oregon State College, La Grande, OR 97850.

Case Three: You are a business person who wants to buy a small computer to automate your bookkeeping.

The criteria for the computer are:

—Availability of a complete package, including hardware, software, and peripherals, that can handle the application
—Sufficient mass storage for records
—Expansion capabilities to handle growth and additional applications
—Local maintenance and service, comparable to the service provided for typewriters and other business equipment

Here, the key is to purchase an off-the-shelf computer that will do the job. An inexpensive system that must be assembled, is unreliable, lacks sufficient storage or adequate peripherals, or does not handle tasks in the way you need them done is no bargain. It will create extra work and expense that will outweigh the initial savings.

The solution is to buy a complete modular computer. You must insist on a demonstration of the entire package so you can be sure it will do the job. A standard maintenance contract should be available. The critical problems will be software, storage, and peripherals—in that order. You will have to deter-

mine exactly what you need and be sure that the computer can handle those needs and provide a margin for expansion. Of course, you must be flexible. The programs, in particular, may not exactly duplicate your manual system. Remember, customizing software is expensive; if the available software is reasonable, you may want to change your methods to conform with it. On the other hand, if the computer's output seems virtually worthless to you, try another vendor; don't listen to claims that a few minor changes will produce a satisfactory system. The computer that you need will represent an investment of $10,000 to $30,000; do not make that investment without specifying your requirements and preparing a test that will tell you whether the computer meets them.

These three case studies demonstrate widely differing criteria, methods, and choices. The engineer is concerned with the computer's electronics—what microprocessor and other devices it uses and what type of electrical connections it provides. The teacher wants an inexpensive computer that is easy to operate and explain, and is concerned with the availability of educational materials and programs for computer-aided-instruction. The business person is concerned with how much the computer can do, how closely it meets his or her requirements, and how reliably it will operate. The programs must perform the required tasks and produce the necessary amount and type of output. It is no wonder that the computers these people should buy differ greatly in appearance, cost, and capabilities.

Clearly the business person faces a more critical decision than does the engineer or the teacher. In the first place, the initial cost of the business computer is much higher, at least on a per-unit basis. The business computer will probably cost $10,000 to $30,000, whereas a one-board computer costs only a few hundred dollars and a single appliance computer costs less than $1,000. Second, the potential loss from buying the wrong computer is far larger for the business person than for the engineer or the teacher. A computer that is unreliable or cannot do the job could be very expensive for a business. An engineer or teacher does not depend as much on the computer and is unlikely to be affected much by a bad choice. Finally, since the tasks are more complex and the requirements more specific, a poorly chosen business computer is far more likely to fail completely than are computers intended primarily for education. The engineer will surely learn something from his or her computer, and the teacher and his or her students will get a tremendous amount of use, education, and enjoyment from their computer, even though neither computer might have been the best possible choice. The business person, however, could end up with lost records, angry suppliers and customers, and disgruntled employees if he or she buys the wrong computer.

Here are some helpful hints.

If you just want to learn about computers, buy an *appliance computer.* They are inexpensive, readily available, easy to use, and intended for beginners.

If you want to use a computer to control something, buy a *one-board computer.* They are inexpensive and easy to connect to external equipment. More expensive computers are often designed to be self-contained.

If you have computer experience, and just want a good system at a reasonable price, buy a *modular computer.* Modular computers can do more than the other types and usually offer better software and hardware.

SUMMARY

The choice of a computer depends on your skills and the way you intend to use the computer. If you are a beginner, you should start with a low-cost computer that you can understand and use. If you plan to depend on the computer for a critical application, you should analyze your needs and make sure that the computer meets them. If you are a knowledgeable user, you should buy a computer that offers the best combination of price, quality, and expansion capabilities. A beginner should find a reliable local dealer who will provide support and encouragement.

A properly selected computer can provide a large amount of profit and enjoyment. New technology will produce low-cost computers that are capable of far more than we can now imagine. But only those who have learned how to use computers will be able to take full advantage of their power.

Appendix 1 Glossary

Access time. The time required to obtain information from a memory.

Accumulator. A register that is used to accumulate or collect results. For example, a computer will typically perform addition by adding a number in memory to the contents of an accumulator; the sum will be left in the accumulator, just as in a pocket calculator.

Acoustic coupler. A device that transforms electrical signals into acoustic signals that can be transmitted over telephone lines.

Address. The identification code that distinguishes one unit of memory from another. The CPU can use the address to select a particular unit.

Addressable cursor. A cursor is addressable if the program can determine where it is on the screen.

Address bus. A bus that the CPU uses to select a particular unit of memory or input/output port for a data transfer.

Addressing methods. The ways to specify the memory addresses the computer needs to execute an instruction. These addresses may tell the computer where to obtain data, where to store data, or where to find the next instruction.

Analog signal. A signal that changes continuously, such as temperature or pressure, rather than in a series of discrete steps, such as time on a clock or the number of people in a room.

APL. A high-level computer language that uses a non-standard set of characters in which special symbols represent operations on sets of data, such as summation, finding the largest element, or determining how many elements there are. APL was developed by Iverson in the early 1960s.

Applications software. The programs that actually do the work (e.g., print invoices, play chess, or teach arithmetic), as opposed to programs that help operate the computer or help programmers write other programs (systems software).

Architecture. Structure of a system; in the case of computers, architecture often refers specifically to the way in which the CPU is organized.

Arithmetic processing unit (APU). A chip that performs arithmetic just like a calculator. No software is necessary, since the APU contains its own programs. A popular APU is the Advanced Micro Devices 9511.

Array. A set of data; an individual element in the set is identified by the name of the set (or the address where the set starts) and its position within the set (or index).

ASCII. American Standard Code for Information Interchange, a 7-bit code often used to represent typed characters in computers.

ASCII keyboard. A keyboard that sends an ASCII character to a computer when the typist presses the corresponding key.

ASR. Asynchronous send/receive; usually refers to a teletypewriter that includes a paper-tape reader and punch, as well as a keyboard and a printer. (See **Asynchronous.**)

Assembler. A computer program that converts assembly language programs (source code) into the numerial form (object code) that the computer actually executes. The assembler converts mnemonic operation codes (such as ADD, SUB, or MOVE) into their numeriucal equivalents, replaces names with their numerical equivalents, and assigns locations in the computer's memory to data and instructions.

Assembly language. A programming language (specific to a particular CPU) in which the programmer uses mnemonic operation codes, names, and labels to refer directly to their numerical equivalents. Assembly language is a low-level language, since the assembler translates each statement directly into one machine language instruction (rather than into a series of machine language instructions as is the case with a high-level language). Writing assembly language programs requires detailed understanding of a particular CPU.

Asynchronous. Operating at irregular time intervals; a typical asynchronous input to a computer is that provided by a person typing on a keyboard.

Automatic line numbering. A feature whereby the computer automatically assigns line numbers to the text entered by the user.

Automatic line renumbering. A feature whereby the computer automatically renumbers lines in a program after the user inserts or deletes lines.

Backplane. The rear of a card cage or other frame, typically containing sockets in which circuit boards, wires, and connectors can be placed.

Bank. A set of registers or memory locations.

BASIC. Beginners' All-purpose Symbolic Instruction Code, a widely used interactive computer language that is especially well-suited to personal computers and beginning users. BASIC was developed by Kemeny and Kurtz in the middle 1960s at Dartmouth College.

Batch. A group of jobs (tasks) that the computer performs one after another. In the batch mode, the computer finishes one job before starting another, thus making short jobs wait for the completion of long jobs that may be ahead of them in line. The alternative is time-sharing in which the computer works on one job for a while, then on another, and so on, giving each one a specific amount of time.

Baud. A measure of the rate at which data is transmitted, expressed in terms of the number of bits (0's or 1's) per second. Common Baud rates are 110, 300, 1200, 2400, 4800, and 9600.

Baudot code. A 5-bit character code used in telegraph systems.

Baud rate generator. A device that determines the rate at which data will be transmitted.

BCD. Binary-coded decimal; a method for representing decimal numbers in which each decimal digit is coded separately.

Benchmark. A problem or program that can be used as a basis for comparing computers, languages, etc.

Bidirectional. Able to transfer data in either direction.

Binary. Number system with base 2; the only digits are 0 and 1. Also refers to any system that has only two possible states or levels, such as a switch that is either on or off.

Bit. A binary digit; it can only have the values 0 or 1.

Blink. A feature on a CRT that lets the computer flash characters on and off. Blinking is often used to show the operator where the cursor is, to call attention to errors, or to emphasize the data that is currently being entered.

Block. A set of data or memory locations.

Block move. Moving an entire set of data from one place in memory to another.

Bootstrap (loader). A program that starts the computer and prepares it to load other programs into its memory. The bootstrap may reside in read-only memory or the operator may have to enter it by hand.

Bottom-up design. A method for designing programs in which the programmer designs, codes, and tests all the parts of the program (or modules) before combining them.

Branch instruction. An instruction that specifically tells the computer where to get the next instruction, thus forcing it to depart from its normal sequential execution. Branch instructions explicitly change the program counter. They may be conditional, that is, only tell the computer where to get the next instruction if a condition is satisfied.

Breakpoint. A condition under which the computer is to stop execution of its current program; used as an aid in debugging programs. The programmer must specify (set) the breakpoints.

Buffer. A temporary storage area for data, often used to hold data that has just been read from an input device or is about to be sent to an output device.

Bundling. Combining hardware, software, and peripherals into a complete system to be sold as a unit.

Bus. A group of parallel electrical connections (wires) that carry signals between devices.

Byte. The basic grouping of bits that a computer can handle at one time, 8 bits in length. One byte can hold one typed ASCII character or two decimal (BCD) digits.

C. A high-level programming language specifically designed for writing systems software, originally developed at Bell Telephone Laboratories.

Card cage. A container that has slots for circuit boards (or cards). The card cage holds the boards in place and provides space for cables and connectors.

Cathode-ray-tube (CRT). The display device that forms the screen in television sets, video monitors, and video terminals.

Character code. A code that assigns numerical values to printed characters. The most popular character code is ASCII (American Standard Code for Information Interchange).

Character generation. The actual formation of characters on a printer or CRT.

Character-oriented editor. An editor that manipulates text as a continuous string of characters rather than as a series of lines. The Electric Pencil from Michael Shrayer Software is an example.

Character printer. A printer that prints one character at a time like a typewriter.

Checksum. A logical sum (no carries) that is often included in a record to guard against recording or transmission errors. The computer checks for errors by recalculating the checksum and comparing it to the value in the record.

Clock. A regular timing signal (i.e., a series of pulses that are equally spaced in time like the ticks of an analog watch) that tells a computer or a component when to do things.

Close (a file). To make a file inactive. The final contents of the file is the last information the user stored in it. The user must generally close a file after working with it.

CMOS. A type of integrated circuit that requires very little power and will withstand a large amount of electrical noise, often used in spacecraft and in portable equipment that must operate from a battery.

COBOL. Common Business-Oriented Language, a popular high-level language that is well-suited to business applications involving complex data records (such as personnel files or customer accounts) and large amounts of printed output.

Coding. The writing of programs in a language that a computer can either execute or translate into an executable form.

Comment. A part of a computer program that explains what is happening to the reader but does not affect how the computer executes the program. Comments are used to document programs.

Compiler. A computer program that translates a program written in a high-level language into a program that a computer can execute. A compiler translates the entire program and produces an executable program (object file).

Composite video. A signal intended for a television set or video monitor that contains both the picture information and the pulses required to keep the picture stable.

Concatenate. To link or join together, place one after another.

Connector. A device that couples one circuit to another.

Controller. A device that manages the operations of a piece of equipment.

CP/M. Control Program/Microcomputer, a widely used disk operating system for personal computers developed by Digital Research (Pacific Grove, CA).

Current-loop interface (or teletype interface). An interface that connects a computer to a standard teletypewriter. Inside the teletypewriter, binary digits (0 or 1) are represented by the presence or absence of current in a loop.

Cursor. A movable visible mark on the surface of a display that indicates where the operator is currently working.

Cyclic redundancy check (CRC). A code that can be added to data records to allow a computer to detect errors in recording or transmission.

Data base. A large collection of well-organized data that is required to perform a task. Typical examples are personnel files, census records, or stock quotations.

Data base management system. A program that allows an operator or other programs to obtain information from a data base or change its contents.

Data bus. A bus that is used to transfer data between the CPU and the memory or the input/output section.

Data file. A file that contains data, rather than programs or procedures.

Data flowchart. A flowchart that shows how a program handles a particular type of data; for example, a data flowchart could show how a payroll program handles overtime or temporary employees.

Data pointer (or pointer). An object that contains the address of a data item rather than the item itself. That is, a pointer tells where the data is located.

Debugger. A program that helps the user find and correct errors in other programs. The errors are called *bugs*.

Decoder. A device that converts coded information into its actual meaning.

Default value. The value that is given to a variable when no value is specifically assigned to it.

Delimiter. A character that marks the end of a data field, as a period marks the end of a sentence.

Diagnostic. A program that checks the operation of a device and reports its findings.

Digital. Having discrete levels rather than continuous values.

Direct memory access (DMA). A method for transferring data to or from a computer's memory without executing a program.

Disassembler. A program that translates a machine language program (object code) back into assembly language, the opposite of an assembler. A disassembler translates numbers into the mnemonic operation codes and names that the programmer can understand more easily.

Disk. A circular device with a magnetic surface that is used to record data. Floppy (flexible) disks are capable of holding thousands of bytes, whereas hard disks are capable of holding millions of bytes.

Disk BASIC. A variation of the BASIC language that is specifically designed for computers that have disks. A disk BASIC will include commands for transferring data and programs to or from the disk. There are many different disk BASICs.

Disk controller. A device that controls the operations of a disk; the interface between a disk and a computer.

Diskette. An individual floppy disk, the recording medium in a floppy disk system. Standard diskettes are 8 inches in diameter whereas mini-diskettes are 5¼ inches in diameter. There is obvious confusion among the terms diskette, disk, and floppy disk, but diskette refers only to the medium, not to the controller or to the unit (player).

Disk operating system (DOS). An operating system that allows the user to transfer programs and data to or from a disk. There are many different (usually incompatible) disk operating systems.

Distributed computing. A computer that has many CPUs rather than one; a computing system that does not depend on a single central computer but may include many computers and intelligent terminals.

Documentation. The techniques used to describe a computer program so that it can be used, maintained, and updated.

Dot-matrix printer. A printer that forms characters from a two-dimensional array of dots. The common sizes for the array are 5×7, 7×7, and 7×9. More dots produce characters that are more distinct and more legible.

Double-density. Special recording method for floppy disks that allows them to store twice as much data as in normal or single-density, allows about 500,000 bytes on one side of an 8-inch disk.

Double-sided. Recording on both sides of a floppy disk.

Dumb terminal. A terminal that has no computing capability of its own, sometimes called a teletype (TTY) replacement or a glass teletype. The user cannot process data on a dumb terminal unless it is connected to a computer.

Dump. A computer program that prints or displays the contents of an entire section of memory or group of registers.

Dynamic debugging package. A program that allows the user to interactively debug his or her program while it is running. Some variations are called dynamic debugging tools (DDT).

Dynamic memory. A memory that loses its contents gradually without any external causes. The contents simply leak away unless the computer periodically rewrites them into the memory; the rewriting is called *refresh.*

Edge connector. A connector that is attached to the edge of a circuit board.

Editor. A computer program that lets the user prepare printed matter (text material) or input for other programs. It allows the user to correct, add, delete, or rearrange material.

Electric Pencil. A popular word processor program developed by Michael Shrayer Software (Palm Springs, CA) for use on personal computers. It is a character-oriented text editor.

Electrostatic printer. A printer that uses a special paper coated with aluminium; it forms characters by burning holes in the aluminium to expose the underlying paper.

EPROM (Erasable PROM). A PROM that can be erased by exposing it to ultraviolet light. Complete erasure usually takes 20 to 30 minutes. (See PROM.)

Executable. Can be executed by a computer without translation.

External reference. The use in a program of a name that is defined in another program.

Field. Part of a record, used to hold a particular piece of information, such as name, rank, address, or job classification.

File. A collection of related records or data treated as a unit. Editors, assemblers, and compilers normally work on program or data files. System commands that the user needs repeatedly may be kept in a *procedure file* to save typing.

Firmware. Programs stored in read-only memory.

Fixed format. Restricted by a particular format or template. The computer will reject as erroneous data that is not in the proper form.

Flag. An indicator that is either on (1) or off (0) and can be used to select between two alternative courses of action. *Semaphore* is another term with the same meaning.

Flexible disk. See **Floppy disk.**

Floating point. A way to represent numbers in which the actual digits are handled separately from the position of the decimal point; for example, 397 may be written as 3.97×100, since the 100 (or 10^2) affects only the position of the decimal point (e.g., 397 and 39.7 differ only in the location of the decimal point). A floating point representation thus requires only the actual digits and the power of 10 by which the digits are to be multiplied.

Floppy disk. A flexible disk with a magnetic surface that can be used to store data. The surface is divided into areas called *sectors*. An IBM-compatible floppy disk uses sectoring and storage techniques originally introduced by IBM. The individual floppy disk is often called a *diskette*.

Flowchart. A graphical representation of a computer program.

Form feed. A feature on some printers that allows the computer to advance a roll of fanfold paper to the top of the next page or form when it has finished printing the previous form.

FORTRAN. Formula Translation Language, a widely used high-level language well suited to problems that can be express in terms of algebraic formulas. FORTRAN was developed in the early 1950s.

Free format. Not restricted by a particular format or template. The user may enter data in an arbitrary manner.

Full-duplex. Capable of both receiving and transmitting at the same time.

Function key. A key that causes a computer to perform a function (such as clearing the screen) or execute a procedure.

General-purpose Interface Bus (GPIB). Hewlett-Packard Bus, or IEEE-488 Bus. A standard interface intended for use in networks of instruments but also used to connect accessories to the Commodore PET computer.

Glitch. A spike or brief burst of noise on an electrical line.

Global. Defined in more than one section of a computer program, a universal variable rather than one used only locally.

Graphics. The use of lines and figures to display information, as opposed to the use of printed characters.

Half-duplex. Capable of receiving or transmitting, but not both at the same time.

Hard copy. Output in a permanent form (usually on paper) rather than shown temporarily on a CRT or visual display.

Hard disk. A disk that is not flexible; more expensive than a flexible (floppy) disk but capable of storing much more data.

Hard error. A permanent error that is caused by a physical fault rather than by transient conditions.

Hardware. The physical equipment that makes up a computer.

Hexadecimal. Number system with base 16, uses the decimal digits (0 through 9) and the letters A through F as its digits.

Hex file. A file consisting of hexadecimal numbers, often the same as **Object File.**

High-level language. A programming language in which the statements represent procedures rather than single computer instructions. A compiler or interpreter translates a program written in a high-level language into a form that the computer can execute. Common high-level languages are BASIC, COBOL, FORTRAN, and PASCAL.

Highlighting. A feature on a display device (CRT) that allows the computer to make some areas brighter than others, used to emphasize important or current information.

Home. Move the cursor to the starting point on the screen.

Impact printer. A printer that forms characters by striking the paper with a character-forming element.

Input/output (I/O) section. The section of the computer that connects the CPU to the peripherals.

Instruction. An input that tells the computer what operation to perform next.

Integrated circuit (IC). A complete electrical circuit on a single chip.

Intelligent (or smart) terminal. A terminal that has its own computing capability and so can be used to process data even when it is not connected to a computer.

Interactive. Capable of carrying on a dialog with the user rather than simply responding to commands.

Interpreter. A computer program that fetches and executes programs written in a high-level language. An interpreter executes each statement immediately after reading it; it does not produce an object program, as a compiler does, so the program must be translated each time it is run.

Interrupt. An input to a computer that forces it to suspend its current program and execute a special program (or service routine). Interrupts are often used to tell the computer that peripherals are ready to send or receive data.

Interrupt-driven. Uses interrupts to interact with the computer.

Interrupt service routine. A program that performs the actions required to respond to an interrupt.

Inverse video. A feature on video terminals that allows the computer to produce white characters on a black background.

I/O driver. A computer program that transfers data to or from an I/O device, also called an *I/O utility.*

I/O utility. A computer program that transfers data to or from an I/O device, also called an *I/O driver.*

Job. A task (program) for a computer to execute.

Joystick. A computer input device that allows the user to move a graphic object (such as a paddle or vehicle) in one or two dimensions, often used in video games.

Jump instruction. An instruction that specifically tells the computer where to get the next instruction, same as branch instruction.

Jumper. A short wire used to connect circuits. The customer usually adds the jumper to select one of several options allowed by the manufacturer.

Jumper selectable. The choice is made by adding the required jumper.

K. 2^{10} or 1024. The term is used to mean 1024 bytes when referring to the size of a computer memory; it is also used to mean 1024 bits when referring to the amount of memory on a single chip.

Kilobit. 1024 bits, also referred to as 1K.

Kilobyte (KB). 1024 bytes, also referred to as 1K.

KSR. Keyboard send /receive; a peripheral that includes both a keyboard and a printer. A KSR model is a complete terminal, as opposed to an RO model that is only a printer.

Label. A name attached to a statement in a program so that other statements can transfer control to that statement or can use the name as if it were a memory address. The label takes the value of the memory address in which the executable version of the statement starts.

Large-scale integration (LSI). Integrated circuits of very great complexity on a single chip. Typical LSI circuits are memory chips, microprocessors, calculator chips, and watch chips.

Latch. A device that retains its contents until new data is specifically entered into it.

Light-emitting diode (LED). A semiconductor device that emits light when a voltage is applied to it in the proper direction. Most LEDs are red, although a few are green or yellow. LEDs are aften used to form displays in pocket calculators and digital watches.

Limit checking. Comparing data items with assigned limits to see if their values are reasonable (i.e., within the limits). Limit checking prevents input errors from producing ridiculous results.

Line editor. An editor that manipulates text as a series of lines rather than as a continuous string of characters.

Line printer. A printer that prints an entire line of characters at a time.

Link editor (or linker). A program that connects a series of programs, resolving references by one to another and assigning all programs to places in memory. A link editor allows the programmer to combine many subprograms into one large program.

Linking loader. A program that combines other programs that may have been assembled or compiled separately and places them all in memory.

Loader. A program that reads another program from an input device into memory.

Local variable. A variable that is only defined in one section of a computer program and can be used in other sections with entirely different meanings.

Logical device. The input or output device to which a program refers. The operating system determines the actual or *physical* device that is to be used; for example, a program may print some output on device #3, which the operating system determines to be a video display. By changing the meaning of device #3 (i.e., by changing the physical device corresponding to logical device #3), the user can make the same program print its output on a printer or on a remote terminal. Varying the correspondence between logical and physical devices is much like switching the output of a stereo amplifier from a front speaker to a back speaker.

Logical file. The file to which a program refers. The operating system determines the actual or *physical* file that is to be used; for example, an editor program may always save the text in the logical file WORKFILE. By changing the physical file to which the logical file WORKFILE refers, the user can have the editor program save the text in the specific file which he or she wants to use.

Logic analyzer. An electronic test instrument that can display the states of many time-varying signals at once.

Logic probe. An electronic instrument that can determine the presence or absence of a pulse at a node.

Lookup table. A table containing all the results of a particular operation organized in a convenient manner. The computer only has to select the proper entry from the table rather than actually performing the operation.

Loop. A self-contained sequence of statements that the computer repeats until a terminating condition is satisfied.

Low-level language. A computer language in which each statement is directly translated into a single machine language instruction.

Machine-independent (or computer independent). Will run on any computer or on any computer that has a compiler or interpreter for a particular language. Also referred to as *portable.*

Machine language. A computer language that consists of instructions written in a numerical form that a computer can execute directly. Also called *machine code* or *object code.*

Macro. A name given to a series of assembly language instructions. The assembler replaces each reference to the macro with a copy of the series of instructions, thus saving a large amount of repetitive typing.

Mainframe. A single large computer such as an IBM 360 or 370, Burroughs 5500 through 7700, Univac 1100, or Control Data 6600.

Maintenance (of programs). Updating and correcting a computer program that is in actual use.

Megabyte. One million bytes.

Memory-mapped input/output. A method for addressing I/O ports whereby they are treated as if they were memory locations.

Memory protect. A feature that stops programs from writing into all or part of memory, thus preserving programs or data from accidental or unauthorized overwriting.

Microprocessor. A single-chip central processing unit; common versions are the Intel 8080 and 8085, MOS Technology 6502, Motorola 6800, and Zilog Z-80.

Microsecond. One millionth of a second.

Mini-disk. A floppy disk that is 5¼ inches in diameter, as opposed to the regular 8-inch size. Also called a *minidiskette* or *minifloppy.*

Mnemonic. Memory jogger, a name that suggests the actual meaning or purpose of the object to which it refers.

Modem. Modulator/demodulator; a device that allows data to be transmitted on a high-frequency communications channel or received from such a channel. The most common channel is the ordinary telephone network.

Modular programming. A programming method in which the programmer divides the entire task into logically separate sections (or **Modules**).

Module. A subprogram or section of a program.

Monitor (hardware). A television-like device (or **Video monitor**) that can be used as an output display.

Monitor (software). A simple operating system that allows the user to enter programs and data into memory, run programs, and observe the contents of registers and memory locations.

MOS. Metal-oxide semiconductor, the most common LSI technology. It is based on field-effect transistors (or FETs).

Motherboard. A board that is only used to provide a base for inserting and connecting other boards.

Multimeter. An electrical instrument that can measure current, voltage, and resistance.

Multitasking. Executing several tasks "at the same time" without having to complete one before starting another. Each task may be given a slice of time or may be executed until its further operation requires the execution of another task or some external action.

Multi-user. Capable of handling more than one user at a time.

Nanosecond. One billionth of a second, abbreviated ns.

Nested subroutines. Subroutines that are used inside other subroutines. The nesting level is the number of transfers of control that are required to reach a particular subroutine without returning.

Nonimpact printer. A printer that forms its characters without striking the paper with a character-forming element. Common types are thermal printers and electrostatic printers.

Nonvolatile memory. A memory that does not lose its contents when power is removed. ROMs and PROMs are nonvolatile.

Number crunching. Computing that involves many complex numerical calculations as opposed to text processing, input/output, or control operations.

Numeric pad. A set of keys that can be used to enter numerical data, usually organized like the keys on an adding machine.

Object file. A file containing instructions that the computer can execute directly.

Octal. Number system with base 8.

Off-line. Not under the control of the computer, occurring without computer intervention. Typical off-line tasks are the punching of cards or paper tape and the production of printed output from tape or disk files.

On-line. Directly under the control of the computer; information is introduced into the computer immediately.

Open (a file). Make a file ready for use. The user generally must open a file before working with it.

Operating system. A computer program that controls the overall operation of a computer and performs such tasks as assigning places in memory to programs and data, processing interrupts, scheduling jobs, and controlling the overall input/output system.

Operation code (or op code). The part of an instruction that tells the computer which operation to perform.

Overlay. The section of a program that is actually in memory at a particular time. A large program can be divided into overlays and run on a computer having a limited amount of memory but having mass storage for the rest of the program.

Page. A subdivision of the memory. Also refers to a screen full of data on a CRT terminal.

Parallel. More than one bit at a time.

Parameter. An item required by a program or subprogram to perform its task.

Parity. A code used to detect recording or transmission errors. Parity is a 1-bit code that makes the total number of one bits in a unit of data, including the parity bit itself, odd *(odd parity)* or even *(even parity).*

PASCAL. A popular high-level language that was developed specifically for use with structured programming and top-down design. PASCAL was developed in the late 1960s by Niklaus Wirth.

Passing parameters. Transferring parameters from a main program to a subroutine.

Peek. A BASIC language command that determines the numerical value stored in a particular memory address. Peek thus allows the BASIC programmer to examine the computer's memory directly, rather than through the BASIC interpreter.

Physical device. An actual input/output device, as opposed to **Logical device.**

Physical file. An actual file residing on a physical device or in memory.

PILOT. A high-level language that is used to develop materials for computer-aided instruction.

Pointer. A storage place that contains the address of a data item rather than the item itself. That is, a pointer tells where the data is located.

Poke. A BASIC language command that stores a specific numerical value in a particular memory address. Poke thus allows the BASIC programmer to change the contents of the computer's memory directly, rather than through the BASIC interpreter.

Procedure file. A file that contains procedural instructions for a computer program, e.g., assignments required by an operating system or some other systems program. A procedure file may, for example, contain all the commands required to assemble or compile a program under a particular operating system.

Program counter. A register that contains the address in memory of the next instruction the computer will execute.

Program library. A collection of debugged and documented programs.

PROM. Programmable read-only memory; a memory that cannot be changed during normal computer operation but that can be programmed under special conditions. Some PROMs (called EPROMs) can be erased with ultraviolet light and reused.

PROM programmer (or PROM burner). A piece of equipment that is used to change the contents of a PROM.

Protected field. An area, usually on a CRT, which the operator cannot change or write over. This feature allows the computer to place the equivalent of a printed form on the screen; the operator can fill in the blank areas but cannot overwrite the text and spacing that makes up the form.

Queue. A set of tasks, storage addresses, or other items that are used in a first-in, first-out manner; that is, the first item entered is the first to be removed.

RAM. Random-access memory, memory that can be both read and altered (written into) during normal operation.

Random-access. Referring to a storage device from which any data item can be removed in the same amount of time. A disk or a computer memory is random-access, whereas a tape is not.

Real-time. In synchronization with the actual occurrence of events.

Real-time clock. A device that interrupts a CPU on a regular basis, such as once per second. By counting the number of such interrupts that have occurred, the computer can keep track of elapsed time.

Record. A collection of related data items. Records usually consist of several fields; in turn, a file usually contains several records.

Refresh. The process of restoring the contents of a **Dynamic memory** before they are lost.

Register. A storage location used to hold data inside the CPU.

Regulator (or voltage regulator). A device that produces a voltage that is always within a specified tolerance of its average value (e.g., 5V ± 0.1V).

Relative branch. A branch instruction that causes the computer to resume program execution after skipping over a specified number of memory locations. The specified number, called a relative offset, may be negative or positive.

Relocatable. Can be placed in any part of memory without changes, can occupy any set of consecutive memory addresses.

Removable disk. A disk in which the recording medium can be physically removed and replaced.

Report writer. A program that produces output according to a specified reporting form, such as a summary of accounts or a financial statement.

Reset. A signal that forces a computer to enter its startup mode.

Retry. To attempt a procedure over again; often refers to reading data from a tape or disk again after finding an error. Disk controllers often perform one retry automatically before reporting an error.

Reverse video. A feature on a CRT that the computer can use to produce the opposite combination of characters and background from that which is usually employed, i.e., white characters on a black background if black on white is normal, and vice versa.

RF converter (or RF modulator). A device that converts singals from a video interface into signals like those from a television station, thus allowing a home television set to be used as a video display.

RO. Receive-only, often refers to a printer without an associated keyboard (a KSR model would have the keyboard).

Rollover. Resolving simultaneous key closures so that they are registered consecutively.

ROM. Read-only memory; a memory containing fixed data that is permanently defined as part of the manufacturing process. A computer can use the information in the ROM, but cannot change it.

RS-232. A standard serial interface defined by the Electronic Industries Association (EIA).

S-100 bus. A standard interface, consisting of 100 parallel connections, used to connect circuit boards inside personal computers.

Screen move. Moving an entire screen's worth of data. A format that can be placed on the screen (often representing a printed form) is referred to as a *screen.*

Scrolling. Moving the information shown on a display up or down so that new information appears at one edge while the old information disappears at the opposite edge.

Self-test. A procedure whereby a program or peripheral checks its own operation.

Serial. One bit at a time.

Serial-access (or sequential-access). Refers to devices from which data can be retrieved only by passing through all locations between the one currently being accessed and the desired one. A tape is a typical serial-access device.

Single-density. Normal recording density for floppy disks; allows about 250,000 bytes on an 8-inch disk.

Single-sided. Recording on only one side of a floppy disk.

Single-user. Capable of handling only one user at at time.

Smart peripheral. A peripheral that has its own internal computer so that it can perform some of the functions normally performed by a central computer or by an interface.

Smart terminal. A terminal that has computing capability of its own and is not totally dependent on the computer to which it is attached.

Socket. A device intended to hold an integrated circuit and connect it to a system electrically. The integrated circuit can be easily removed from the socket, thus simplifying testing and replacement.

Soft error. An error that is temporary in nature and is caused by transient conditions.

Software. Computer programs.

Source file. A file that contains statements written in a computer language. A source file must be assembled, compiled, or interpreted before it can be executed.

Spooling. Saving output data in a disk or tape file and sending it to the actual output device at a later time. Spooling keeps the computer from having to wait for a slow output device to print.

Stack. A set of tasks, storage addresses, or other items that are used in a last-in, first-out manner; that is, the last item entered is the first to be removed.

Standard teletypewriter. A teletypewriter that prints at a rate of 10 characters per second.

Static memory. A memory that does not lose its contents without external causes, as opposed to a dynamic memory.

String. An array (set of data) consisting of characters.

String functions. Procedures that allow the programmer to operate on data consisting of characters rather than numbers. Typical functions are insertion, deletion, concatenation, search, and replacement.

Structured programming. A programming method in which all programs consist of structures from a limited but complete set; each structure has a single entry and a single exit to simplify debugging.

Stub. A stand-in for a program that has not yet been written; it should produce the same incidental effects as the program it replaces.

Subroutine. A program that is subordinate to another program. The process whereby the computer transfers control to the subroutine is referred to as a *subroutine call* and the mechanism that is used to transfer control to and from the subroutine is referred to as a *subroutine linkage.*

Supervisor. A program that oversees the execution of other programs. The operating system is the overall supervisor for a computer, but user programs may have their own individual supervisors that control the execution of subroutines.

Symbol table. A table defining all the names and labels used in a program.

Synchronization. A means of insuring that transmitter and receiver are operating together (in phase or in lockstep).

Synchronization (or sync) character. A character that is used only to provide synchronization.

Synchronous. Operating at regular time intervals according to a clock.

Syntax. The rules governing the structure of a language. The language may be either human (natural) or a computer language.

Systems program. A program that is part of the operating system.

Systems software. Programs that perform administrative functions or aid in the development of other programs but do not actually perform any of the computer's ultimate workload.

Tabbing. Moving the cursor horizontally to a specified position on the screen, much like using tabs on a typewriter.

Task. A program that the computer can execute.

Task status. The current state of a task, the information that must be saved if the task is to be interrupted and restarted.

Task swapping. Changing active tasks, suspending one task and beginning another. The suspended task is often said to be *rolled out* and the newly started task to be *rolled in.*

Teletypewriter (TTY). A peripheral containing a keyboard and a character printer; the most popular version is the ASR-33 model produced by Teletype Corporation of Skokie, IL.

Terminal. An input/output device used to enter data into a computer and record the output; usually consists of a keyboard and a printer or video display.

Thermal printer. A printer that forms characters by applying heat to special heat-sensitive paper.

Time-sharing. Providing service to many users by working on each one's task part of the time.

Top-down design. A method for designing computer programs in which the programmer designs, codes, and tests the overall structure first and subsequently defines parts of the structure in increasingly greater detail. The programmer must replace subprograms that have not yet been written with stubs.

Top-of-form. A feature that causes a printer or CRT to skip to the top of the next form.

Trace. A debugging aid that provides information about a program while it is executing. The trace prints all or some of the intermediate results.

Tractor feed. A mechanical device used to move paper through a printer. The roller that moves the paper has sprockets on each end that fit into fanfold paper that has a matching pattern of holes.

TTL. Transistor-transistor logic, the most widely used type of integrated circuit.

Turnkey system. A computer that is ready to be used without adding any hardware or software; it is complete as packaged for a particular application.

Unbundling. Separating hardware, software, and peripherals so that they can be purchased individually.

Unprotect. Remove characters from write protection and allow them to be overwritten.

User program. A program that performs a task specified by the ultimate user of the computer.

User-programmable. The user can determine the purpose of the feature by writing an appropriate program.

Utility program. A program that performs a basic systems task, such as loading and saving programs, initiating program execution, observing and changing the contents of memory locations, or converting programs from one form to another.

Vectored interrupt. An interrupt that directs the computer to the specific service routine required to respond to it.

Video board. An interface between a computer and a video display.

Video monitor. A television-like display device, differs from a normal television set in that it cannot receive television signals over the air.

Video RAM. Separate memory that is used to hold information being shown on a video display.

Video terminal. A terminal that displays data on a CRT.

Volatile memory. A memory that loses its contents when power is removed. Most computer memory is volatile.

Winchester disk. A type of hard disk.

Word. The characteristic bit length of a computer, usually the length of its data paths, registers, and arithmetic unit.

Word processing system. A system that processes text, performing such functions as paragraphing, paging, left and right justification, and rearrangement of lines.

Workspace. An area of memory or backup storage assigned for temporary use. Also called a *workfile.*

Write protect. A feature that prevents the computer from writing into an area of memory or storage. This protects the area from accidental overwriting or unauthorized changes.

Appendix 2 Codes

Table A2-1. Hexadecimal to Decimal Conversion Table

Hexadecimal Digit	Decimal Value	Binary Value
0	0	0000
1	1	0001
2	2	0010
3	3	0011
4	4	0100
5	5	0101
6	6	0110
7	7	0111
8	8	1000
9	9	1001
A	10	1010
B	11	1011
C	12	1100
D	13	1101
E	14	1110
F	15	1111

Table A2-2. ASCII Character Code

BITS				B_7	0	0	0	0	1	1	1	1
				B_6	0	0	1	1	0	0	1	1
				B_5	0	1	0	1	0	1	0	1
				Column →	0	1	2	3	4	5	6	7
B_4	B_3	B_2	B_1	Row ↓								
0	0	0	0	0	NUL	DLE	SP	0	@	P	‘	p
0	0	0	1	1	SOH	DC1	!	1	A	Q	a	q
0	0	1	0	2	STX	DC2	”	2	B	R	b	r
0	0	1	1	3	ETX	DC3	#	3	C	S	c	s
0	1	0	0	4	EOT	DC4	$	4	D	T	d	t
0	1	0	1	5	ENQ	NAK	%	5	E	U	e	u
0	1	1	0	6	ACK	SYN	&	6	F	V	f	v
0	1	1	1	7	BEL	ETB	’	7	G	W	g	w
1	0	0	0	8	BS	CAN	(	8	H	X	h	x
1	0	0	1	9	HT	EM	)	9	I	Y	i	y
1	0	1	0	A	LF	SUB	*	:	J	Z	j	z
1	0	1	1	B	VT	ESC	+	;	K	[	k	{
1	1	0	0	C	FF	FS	,	<	L	\	l	!
1	1	0	1	D	CR	GS	—	=	M	]	m	}
1	1	1	0	E	SO	RS	.	>	N	⌒	n	~
1	1	1	1	F	SI	US	/	?	O	___	o	DEL

To form the code for any character on the chart, first locate the character on the chart. The most significant hex digit of the code is the column number, and the lease significant digit is the row number.

Examples: The code for A = 41
* = 2A

Legend:

NUL	Null
SOH	Start of Heading
STX	Start of Text
ETX	End of Text
EOT	End of Transmission
ENQ	Enquiry
ACK	Acknowledge
BEL	Bell
BS	Backspace
HT	Horizontal Tab
LF	Line Feed
VT	Vertical Tab
FF	Form Feed
CR	Carriage Return
SO	Shift Out
SI	Shift In
DLE	Data Link Escape
DC1	Device Control 1
DC2	Device Control 2
DC3	Device Control 3
DC4	Device Control 4 (Stop)
NAK	Negative Acknowledge
SYN	Synchronous Idle
ETB	End of Transmission Block
CAN	Cancel
EM	End of Medium
SUB	Substitute
ESC	Escape
FS	File Separator
GS	Group Separator
RS	Record Separator
US	Unit Separator
DEL	Delete

Appendix 3

Simple Electrical Components

RESISTORS

Resistors impede the flow of current through a circuit. The amount of resistance is indicated on standard resistors by four color bands. If we place the resistor so that the bands are at the left (Figure A3-1), then the meaning of the bands from left to right is

- Bands 1 and 2 are the first and second digits of the resistance in ohms.
- Band 3 is the number of zeros that follow the second digit.
- Band 4 is the tolerance; if this band is missing, the tolerance (i.e., the precision to which the resistance is known) is 20%.

The color codes are

0	Black
1	Brown
2	Red
3	Orange
4	Yellow
5	Green
6	Blue
7	Violet
8	Gray
9	White
0.1	Gold
0.01	Silver

The Tolerance codes are

5%	Gold
10%	Silver

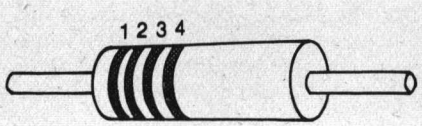


Figure A3-1. A Standard Resistor with Color Bands.

CAPACITORS

Capacitors store electrical charge and thus reduce the effects of changes in voltage levels. Capacitance is measured in units of farads; typical values range from 1 picofarad (*pf:* a picofarad is one millionth of a millionth of a farad or 10^{-12} farad) to 100 microfarads (μf: a microfarad is one millionth of a farad or 10^{-6} farad). Large capacitors must be connected in the correct direction; the polarity is indicated by a band or "+".

DIODES

Diodes allow the flow of current in one direction only. A type number on the body describes the diode, such as a 1N728. Diodes come in a variety of sizes. Usually a band at one end indicates the cathode. Diodes will only conduct current when the cathode (negative terminal) is at a lower voltage than the anode (positive terminal).

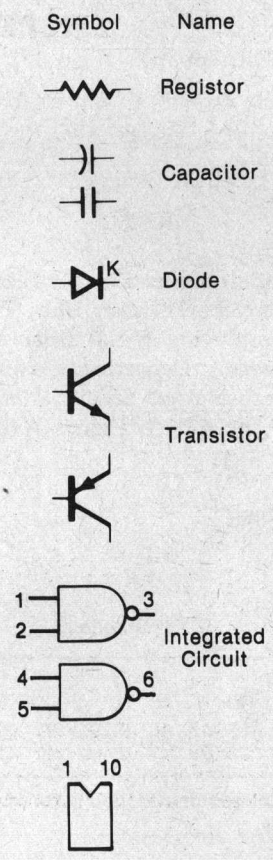
TRANSISTORS

Transistors act as switches, amplifiers, or oscillators. A type number on the body describes the transistor, such as a 2N3904.

INTEGRATED CIRCUITS

Integrated circuits are single chips that contain resistors, capacitors, diodes, transistors and other circuits. The most popular family of integrated circuits is TTL or transistor-transistor logic; the next most popular technology is complementary metal-oxide-semiconductor or CMOS. The most widely used series of integrated circuits is the 7400 series originated by Texas Instruments (but now supplied by many manufacturers). Variations of the primary 7400 series are 74LS (for low-power Schottky) and 74S (for Schottky — a high-speed process).

Table A3-1. Components and Their Symbols.



REFERENCES

Boylestad, R. L. *Introductory Circuit Analysis.* Columbus, OH: Merrill, 1974.

Johnson, D. E., et al. *Basic Electric Circuit Analysis.* Englewood Cliffs, NJ: Prentice-Hall, 1978.

NCR Basic Electronics Course. Indianapolis, IN: Sams, 1978.

Rony, P. R., and Larsen, D. G. *Logic and Memory Experiments Using TTL Integrated Circuits.* Indianapolis, IN: Sams, 1978.

Senturia, S. D., and Wedlock, B. D. *Electronic Circuits and Applications.* New York: Wiley, 1975.

Microprocessors, memories, and other LSI circuits commonly found in personal computers are fully described from the engineering point of view in

Osborne, A. *An Introduction to Microcomputers. Volume 2: Some Real Microprocessors.* Berkeley, CA: Osborne/McGraw-Hill, 1978.

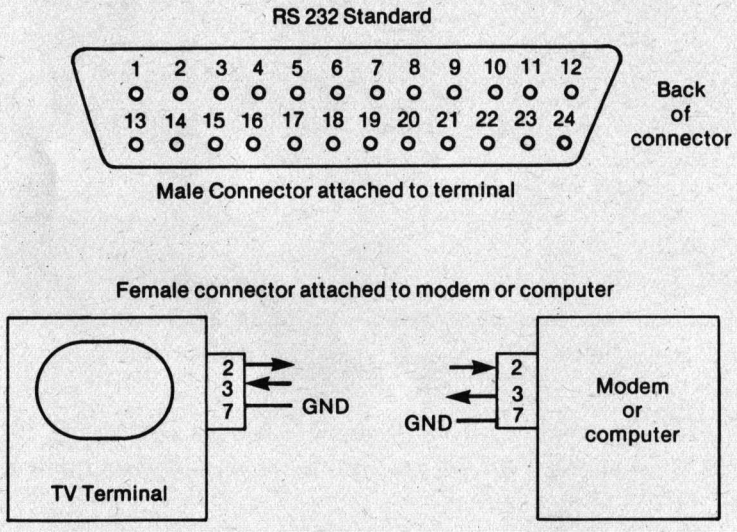
Osborne, A. *An Introduction to Microcomputers. Volume 3: Some Real Support Devices.* Berkeley, CA: Osborne/McGraw-Hill, 1978.

Appendix 4 Interface Standards

RS-232

RS-232 is a serial interface standard defined by the Electronic Industries Association. The specific document is "Interface between Data Terminal Equipment and Data Communication Equipment Employing Serial Binary Data Exchange", Electronic Industries Association, Engineering Department, 2001 Eye St. N.W., Washington, D.C. 20006. An elementary discussion of RS-232 is in G. Pickles, "Who's Afraid of RS-232", *Kilobaud*, May 1977, pp. 50-54. Figure A4-1 shows the standard RS-232 connector and lists the signals.

Figure A4-1. RS-232 Connections.



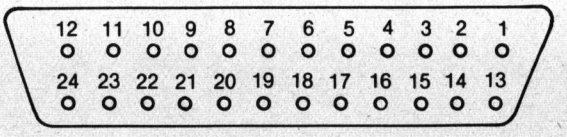
RS-232 SIGNALS

CODE	PIN NUMBER	SIGNAL NAME
AA	1	PROTECTIVE GROUND
BA	2	DATA TRANSMITTED FROM SENDER
BB	3	DATA RECEIVED FROM SENDER
CA	4	REQUEST TO SEND (FROM SENDER)
CB	5	CLEAR TO SEND (TO SENDER)
CC	6	DATA SET READY
AB	7	SIGNAL GROUND
CF	8	CARRIER DETECT
	9-14	UNUSED
DB	15	TRANSMITTER CLOCK (INTERNAL)
	16	UNUSED
DD	17	RECEIVER CLOCK
	18-19	UNUSED
CD	20	DATA TERMINAL READY
	21	UNUSED
CE	22	RING INDICATOR
	23	UNUSED
DA	24	TRANSMITTER BIT CLOCK (EXTERNAL)
	25	UNUSED

IEEE-488

IEEE-488 is a parallel interface standard defined by the Institute of Electrical and Electronic Engineers. The specific document is "IEEE Standard Digital Interface for Programmable Instrumentation", The Institute of Electrical and Electronic Engineers, inc., 345 East 47th Street, New York, NY 10017. The IEEE-488 bus is also discussed in C. A. Ogdin, "Microcomputer Buses—Part II", *Mini-Micro Systems,* July 1978, pp. 76–80 and in S. C. Baunach, "An Example of an M6800-Based GPIB Interface", *EDN,* September 20, 1977, pp. 125–128. Figure A4-2 shows the standard IEEE-488 connector and lists the signals.

Figure A4-2. IEEE-488 Connections.



PIN NUMBER	SIGNAL NAME	MEANING
1	DATA I/O	Bit 1 of data
2	DATA I/O	Bit 2 of data
3	DATA I/O	Bit 3 of data
4	DATA I/O	Bit 4 of data
5	END OR IDENTIFY (EOI)	Indicates the end of a data transfer or requests sources with data to identify themselves
6	DATA VALID (DAV)	Signifies that valid information is available on the data lines
7	NOT READY FOR DATA (NRFD)	Signifies that the receiver has accepted the data and is ready for more
8	NOT DATA ACCEPTED (NDAC)	Signifies that the receiver is ready to accept data
9	INTERFACE CLEAR (IFC)	Places system in a known state
10	SERVICE REQUEST (SRQ)	Signifies a request to send data
11	ATTENTION	indicates nature of information on data lines
12	SHIELD	
13	DATA I/O	Bit 5 of data
14	DATA I/O	Bit 6 of data
15	DATA I/O	Bit 7 of data
16	DATA I/O	Bit 8 of data
17	REMOTE ENABLE (REN)	Commands receivers to select remote operation
18-24	GROUND	

CENTRONICS (PRINTER) INTERFACE

The Centronics parallel interface (see Figure A4-3) is a 36-line interface often used to connect printers to computers. There is no formal definition for this standard, nor is there a sponsoring body. However, it is widely used (Centronics Data Computer Corporation of Hudson, NH is a large manufacturer of printers).

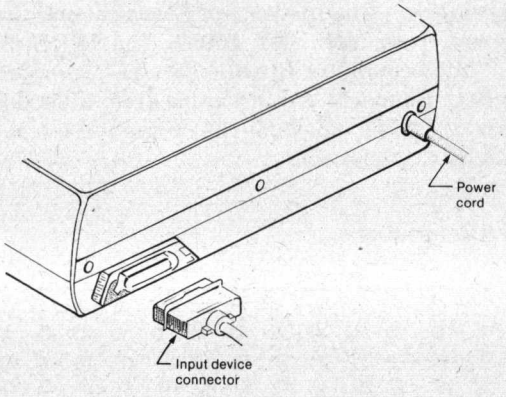


Figure A4-3. Standard Parallel Interface Connection. For the standard parallel interface connection, connect the 36-pin input device connector to the connector at the rear of the printer, as shown here.

INTERFACE PIN OUTS

PIN	SIGNAL
1	DATA STROBE
2	DATA BIT 1
3	DATA BIT 2
4	DATA BIT 3
5	DATA BIT 4
6	DATA BIT 5
7	DATA BIT 6
8	DATA BIT 7
9	—
10	ACKNOWLEDGE
11	BUSY
12	PAPER OUT or PRINT OFF
13	—
14	GROUND
15	—
16	GROUND
17	CHASSIS GROUND
18	+5V
19	—
20	—
21	—
22	—
23	—
24	—
25	— TWISTED PAIR GROUND
26	—
27	—
28	—
29	—
30	—
31	—
32	—
33	GROUND
34	—
35	—
36	—

S-100 BUS

The S-100 bus is a parallel interface consisting of 100 lines. Table A4-1 lists the signals.

References:

Elmquist, K. A., et al. "Standard Specifications for S-100 Bus Interface Devices," *Computer,* July 1979, pp. 28–51.

Osborne, A. *An Introduction to Microcomputers. Volume 3: Some Real Support Devices.* Berkeley, CA: Osborne/McGraw-Hill, 1978, pp. J4–1 through J4–9.

Smith, M. L. "Build Your Own Interface," *Kilobaud,* June 1977, pp. 22–28.

The magazine *S-100 Microsystems* (P.O. Box 1192, Mountainside, NJ 07092) is entirely dedicated to computer systems using the S-100 bus.

HEATH H8 BUS

The Heath H8 bus is a 50-line parallel interface defined for the Heath H8 computer from Heath Corporation of Benton Harbor, Michigan. Table A4-2 lists the signals.

RADIO SHACK BUS

The Radio Shack bus is a 40-line parallel interface defined for the Radio Shack TRS-80 microcomputer from Radio Shack Division of Tandy Corporation of Fort Worth, Texas. Table A4-3 lists the signals.

SS-50 BUS

The SS-50 bus is a 50-line parallel interface originated by Southwest Technical Products Corporation of San Antonio, Texas. It is mostly used with computers based on the Motorola 6800 microprocessor. Table A4-4 lists the signals.

APPLE I/O BUS

The Apple I/O bus is a 50-line parallel interface defined for the Apple II microcomputer from Apple Computer Corporation of Cupertino, California. Table A4-5 lists the signals.

KIM BUS

The KIM expansion bus is a 44-line parallel interface defined for the KIM-1 microcomputer produced by MOS Technology of Norristown, PA. Table A4-6 lists the signals.

OSI-48 BUS

The Ohio Scientific Instruments OSI-48 bus is a 48-line parallel interface originated by Ohio Scientific Instruments of Aurora, Ohio. Table A4-7 lists the signals.

Table A4-1. The S-100 Bus

PIN	SIGNAL
1	+ 8 V unregulated (input to + 5V regulators)
2	+ 16 V unregulated (input to + 12V regulators)
3	EXTERNAL READY (XRDY)
4	VECTORED INTERRUPT REQUEST 0 (VI0)
5	VECTORED INTERRUPT REQUEST 1 (VI1)
6	VECTORED INTERRUPT REQUEST 2 (VI2)
7	VECTORED INTERRUPT REQUEST 3 (VI3)
8	VECTORED INTERRUPT REQUEST 4 (VI4)
9	VECTORED INTERRUPT REQUEST 5 (VI5)
10	VECTORED INTERRUPT REQUEST 6 (VI6)
11	VECTORED INTERRUPT REQUEST 7 (VI7)
12	NOT USED
13	NOT USED
14	NOT USED
15	NOT USED
16	NOT USED
17	NOT USED
18	STATUS BUFFER DISABLE (STADSB')
19	COMMAND/CONTROL BUFFER DISABLE (C/CDSB')
20	UNPROTECT MEMORY (UNPROT)
21	SINGLE STEP (SS)
22	ADDRESS BUFFER DISABLE (ADDR DSBL')
23	DATA OUTPUT BUFFER DISABLE (DO DSBL)
24	PHASE 2 TTL LEVEL CLOCK (ϕ 2)
25	PHASE 1 TTL LEVEL CLOCK (ϕ 1)
26	HOLD ACKNOWLEDGE (PHLDA)
27	PROCESSOR WAIT ACKNOWLEDGE (PWAIT)
28	PROCESSOR INTERRUPT ENABLE (PINTE)
29	ADDRESS BIT 5 (A5)
30	ADDRESS BIT 4 (A4)
31	ADDRESS BIT 3 (A3)
32	ADDRESS BIT 15 (A15)
33	ADDRESS BIT 12 (A12)
34	ADDRESS BIT 1 (A1)
35	DATA OUTPUT BIT 1 (DO1)
36	DATA OUTPUT BIT 0 (DO0)
37	ADDRESS BIT 10 (A10)
38	DATA OUTPUT BIT 4 (DO4)
39	DATA OUTPUT BIT 5 (DO5)
40	DATA OUTPUT BIT 6 (DO6)
41	DATA INPUT BIT 2 (DI2)
42	DATA INPUT BIT 3 (DI3)
43	DATA INPUT BIT 7 (DI7)
44	M1 STATUS (SM1) — INDICATES FIRST CYCLE OF AN INSTRUCTION
45	STATUS OUTPUT (SOUT)
46	STATUS INPUT (SINP)
47	STATUS MEMORY READ (SMEMR)
48	STATUS HALT ACKNOWLEDGE (SHLTA)
49	CLOCK SIGNAL (CLOCK')
50	GROUND

Table A4-2 (Continued)

51	+ 8 V unregulated (input to + 5V regulators)
52	+ 16 V unregulated (input to + 12V regulators)
53	UNUSED
54	EXTERNAL CLEAR (EXT CLR')
55	UNUSED
56	UNUSED
57	UNUSED
58	UNUSED
59	UNUSED
60	UNUSED
61	UNUSED
62	UNUSED
63	UNUSED
64	UNUSED
65	UNUSED
66	UNUSED
67	UNUSED
68	MEMORY WRITE (MWRITE)
69	UNUSED
70	PROTECT (PROT)
71	RUN
72	READY (PRDY)
73	INTERRUPT REQUEST (PINT')
74	PROCESSOR HOLD (PHOLD')
75	PROCESSOR RESET (PRESET')
76	SYNCHRONIZATION (PSYNC)
77	WRITE ENABLE (PWR')
78	DATA BUS IN (PDBIN)
79	ADDRESS BIT 0 (A0)
80	ADDRESS BIT 1 (A1)
81	ADDRESS BIT 2 (A2)
82	ADDRESS BIT 6 (A6)
83	ADDRESS BIT 7 (A7)
84	ADDRESS BIT 8 (A8)
85	ADDRESS BIT 13 (A13)
86	ADDRESS BIT 14 (A14)
87	ADDRESS BIT 11 (A11)
88	DATA OUTPUT BIT 2 (DO2)
89	DATA OUTPUT BIT 3 (DO3)
90	DATA OUTPUT BIT 7 (DO7)
91	DATA INPUT BIT 4 (DI4)
92	DATA INPUT BIT 5 (DI5)
93	DATA INPUT BIT 6 (DI6)
94	DATA INPUT BIT 1 (DI1)
95	DATA INPUT BIT 0 (DI0)
96	STATUS INTERRUPT ACKNOWLEDGE (SINTA)
97	STATUS WRITE OUTPUT (SWO)
98	STATUS STACK (SSTACK)
99	POWER ON CLEAR (POC')
100	GROUND

Table A4-2. The Heath H8 Bus

PIN	SIGNAL
0	GROUND
1	GROUND
2	– 18V UNREGULATED
3	INTERRUPT LEVEL 3 (INT3')
4	INTERRUPT LEVEL 4 (INT4')
5	INTERRUPT LEVEL 5 (INT5')
6	INTERRUPT LEVEL 6 (INT6')
7	INTERRUPT LEVEL 7 (INT7')
8	INTERRUPT LEVEL 1 (INT1')
9	INTERRUPT LEVEL 2 (INT2')
10	DATA BIT 0 (D0')
11	DATA BIT 1 (D1')
12	DATA BIT 2 (D2')
13	DATA BIT 3 (D3')
14	DATA BIT 4 (D4')
15	DATA BIT 5 (D5')
16	DATA BIT 6 (D6')
17	DATA BIT 7 (D7')
18	UNUSED
19	MACHINE CYCLE 1 (M1)
20	READY IN (RDYIN)
21	INPUT/OUTPUT WRITE (I/OW)
22	CLOCK PHASE 2 (ϕ 2')
23	MEMORY WRITE (MEMW)
24	GROUND
25	HOLD ACKNOWLEDGE (HLDA)
26	INPUT/OUTPUT READ (I/OR)
27	HOLD
28	MEMORY READ (MEMR)
29	RESET (RESET')
30	ADDRESS BIT 0 (A0')
31	ADDRESS BIT 1 (A1')
32	ADDRESS BIT 2 (A2')
33	ADDRESS BIT 3 (A3')
34	ADDRESS BIT 4 (A4')
35	ADDRESS BIT 5 (A5')
36	ADDRESS BIT 6 (A6')
37	ADDRESS BIT 7 (A7')
38	ADDRESS BIT 8 (A8')
39	ADDRESS BIT 9 (A9')
40	ADDRESS BIT 10 (A10')
41	ADDRESS BIT 11 (A11')
42	ADDRESS BIT 12 (A12')
43	ADDRESS BIT 13 (A13')
44	ADDRESS BIT 14 (A14')
45	ADDRESS BIT 15 (A15')
46	ROM DISABLE'
47	+ 18V UNREGULATED
48	+ 18V UNREGULATED
49	+ 8V UNREGULATED

Table A4-3. The Radio Shack 40 Line Bus

PIN NUMBER	SIGNAL NAME	DESCRIPTION
1	RAS*	Row Address Strobe Output for 16-Pin Dynamic RAMs
2	SYSRES*	System Reset Output, Low During Power Up Initialize or Reset Depressed
3	CAS*	Column Address Strobe Output for 16-Pin Dynamic RAMs
4	A10	Address Output
5	A12	Address Output
6	A13	Address Output
7	A15	Address Output
8	GND	Signal Ground
9	A11	Address Output
10	A14	Address Output
11	A8	Address Output
12	OUT*	Peripheral Write Strobe Output
13	WR*	Memory Write Strobe Output
14	INTAK*	Interrupt Acknowledge Output
15	RD*	Memory Read Strobe Output
16	MUX	Multiplexor Control Output for 16-Pin Dynamic RAMs
17	A9	Address Output
18	D4	Bidirectional Data Bus
19	IN*	Peripheral Read Strobe Output
20	D7	Bidirectional Data Bus
21	INT*	Interrupt Input (Maskable)
22	D1	Bidirectional Data Bus
23	TEST*	A Logic "0" on TEST* Input Tri-States A0-A15, DO-D7, WR*, RD*, IN*, OUT*, RAS* CAS*, MUX*
24	D6	Bidirectional Data Bus
25	A0	Address Output
26	D3	Bidirectional Data Bus
27	A1	Address Output
28	D5	Bidirectional Data Bus
29	GND	Signal Ground
30	D0	Bidirectional Data Bus
31	A4	Address Output
32	D2	Bidirectional Data Bus
33	WAIT*	Processor Wait Input, to Allow for Slow Memory
34	A3	Address Output
35	A5	Address Output
36	A7	Address Output
37	GND	Signal Ground
38	A6	Address Output
39	+5V	5 Volt Output (Limited Current)
40	A2	Address Output

NOTE: * means Negative (Logic "0") True Input or Output.

Mates with AMP P/N 88103-1 Card Edge Connector or Equivalent.

Table A4-4. The SS-50 (SWTP) Bus

LINE	FUNCTION
D0-D7 (inverted)	Inverted data bits 0 through 7
A0-A15	Address bits 0 through 15
GND	Common ground
7-8 VDC UNREG	+7 or +8 V unregulated power
−12	−12 V power
+12	+12 V power
INDEX	Unused line, prevents the circuit boards from being plugged on incorrectly
M.RESET	Manual reset
NMI*	Non-maskable interrupt
IRQ*	Interrupt request
UD1, UD2	User-defined lines
ϕ2	Phase 2 of system clock
VMA*	Valid memory address
R/W	Read/write
RESET*	Reset
BA	Bus available
HALT*	Halt
ϕ1	Phase 1 of system clock
110b	Clock for 110 Baud communications
150b	Clock for 150 Baud communications
300b	Clock for 300 Baud communications
600b	Clock for 600 Baud communications
1200b	Clock for 1200 Baud communications

Table A4-5. The Apple I/O Bus

PIN	SIGNAL
1	I/O ENABLE'
2	ADDRESS BIT 0 (AD0)
3	ADDRESS BIT 1 (AD1)
4	ADDRESS BIT 2 (AD2)
5	ADDRESS BIT 3 (AD3)
6	ADDRESS BIT 4 (AD4)
7	ADDRESS BIT 5 (AD5)
8	ADDRESS BIT 6 (AD6)
9	ADDRESS BIT 7 (AD7)
10	ADDRESS BIT 8 (AD8)
11	ADDRESS BIT 9 (AD9)
12	ADDRESS BIT 10 (AD10)
13	ADDRESS BIT 11 (AD11)
14	ADDRESS BIT 12 (AD12)
15	ADDRESS BIT 13 (AD13)
16	ADDRESS BIT 14 (AD14)
17	ADDRESS BIT 15 (AD15)
18	READ/WRITE (R/W')
19	UNUSED
20	I/O SELECT'
21	READY (RDY)
22	DIRECT MEMORY ACCESS (DMA')
23	INTERRUPT DAISY CHAIN OUTPUT (INTDAISYOUT)
24	DIRECT MEMORY ACCESS DAISY CHAIN OUTPUT (DMADAISYOUT)
25	+ 5V REGULATED
26	GROUND
27	DIRECT MEMORY ACCESS DAISY CHAIN INPUT (DMADAISYIN)
28	INTERRUPT DAISY CHAIN INPUT (INTDAISYIN)
29	NONMASKABLE INTERRUPT (NMI')
30	INTERRUPT REQUEST (IRQ')
31	RESET (RES')
32	INHIBIT (INH')
33	– 12V REGULATED
34	– 5V REGULATED
35	UNUSED
36	7 MEGAHERTZ CLOCK (7MHz)
37	CLOCK PHASE 3 (ϕ 3) — 2MHz
38	CLOCK PHASE 1 (ϕ 1) — 1MHz
39	USER1
40	CLOCK PHASE 0 (ϕ 0) — 1MHz
41	DEVICE ENABLE (DEV ENAB)
42	DATA BIT 7 (D7)
43	DATA BIT 6 (D6)
44	DATA BIT 5 (D5)
45	DATA BIT 4 (D4)
46	DATA BIT 3 (D3)
47	DATA BIT 2 (D2)
48	DATA BIT 1 (D1)
49	DATA BIT 0 (D0)
50	+ 12V REGULATED

Table A4-6. The KIM Expansion Bus

PIN	SIGNAL
1	SYNCHRONIZATION (SYNC)
2	READY (RDY)
3	CLOCK PHASE 1 (ϕ 1)
4	INTERRUPT REQUEST (IRQ)
5	SET OVERFLOW (SO)
6	NONMASKABLE INTERRUPT (NMI)
7	RESET (RST)
8	DATA BIT 7 (DB7)
9	DATA BIT 6 (DB6)
10	DATA BIT 5 (DB5)
11	DATA BIT 4 (DB4)
12	DATA BIT 3 (DB3)
13	DATA BIT 2 (DB2)
14	DATA BIT 1 (DB1)
15	DATA BIT 0 (DB0)
16	DECODER OUTPUT 6 (K6 — SELECTS 0600-06FF)
17	SINGLE STEP OUTPUT (SST OUT)
18-20	UNUSED
21	+ 5 VOLTS
22	GROUND
A	ADDRESS BIT 0 (AB0)
B	ADDRESS BIT 1 (AB1)
C	ADDRESS BIT 2 (AB2)
D	ADDRESS BIT 3 (AB3)
E	ADDRESS BIT 4 (AB4)
F	ADDRESS BIT 5 (AB5)
H	ADDRESS BIT 6 (AB6)
J	ADDRESS BIT 7 (AB7)
K	ADDRESS BIT 8 (AB8)
L	ADDRESS BIT 9 (AB9)
M	ADDRESS BIT 10 (AB10)
N	ADDRESS BIT 11 (AB11)
P	ADDRESS BIT 12 (AB12)
R	ADDRESS BIT 13 (AB13)
S	ADDRESS BIT 14 (AB14)
T	ADDRESS BIT 15 (AB15)
U	CLOCK PHASE 2 (ϕ 2)
V	READ/WRITE (R/W)
W	READ/WRITE' (R/W')
X	PHASE LOCKED LOOP TEST (PLL TEST)
Y	CLOCK PHASE 2' (ϕ 2')
Z	RAM/READ/WRITE (RAM/R/W)

Table A4-7. The OSI 48-Line Bus

PIN	DESCRIPTION
B1	WAIT. When pulled low by a system board, causes processor clock to slow down to speed of approximately 500 kHz. Used to service slow memory and I/O devices.
B2	NMI (non-maskable interrupt). When brought low, a non-blockable interrupt occurs, causing the processor to stop its operation and service this interrupt, that is, go to a specific memory location and execute an interrupt service routine.
B3	IRQ (interrupt request). An interrupt that can be masked by the processor. The processor can choose to ignore this interrupt under program control. If the interrupt is not masked, it will function as NMI above.
B4	DD (data direction). When pulled low by a system board, it changes the data direction of the 8T26 buffers on the CPU board, switching the processor from outputting data to the bus to listening to the bus.
B5-B12	Bidirectional, eight-bit data bus for communication between the processor and system boards.
B13-B16	Upper data bits on some systems.
B17	Optional reset line used to clear all PIAs and similar I/O circuitry in the system.
B18	Spare.
B19-B22	Memory management address lines (the OSI system can address memory in 64K blocks up to at least 768K).
B23	+ 12 volt power connection.
B24	– 9 volt power connection.
B25-B26	+ 5 volt power connection.
B27-B28	Ground.
B29-B38	Ten low-order address lines.
B39	ϕ2. Used to clock external circuits or external I/O interfaces, such as the A/D converter (see a 6502 data sheet for more details).
B40	R/W (read/write). Originates at the microprocessor and specifies read or write operations on the data bus.
B41	VMA (valid memory address). Only used in conjunction with the 6800. The 6502 always has this line high.
B42	ϕ2-VMA. Master timing signal for enabling memory and I/O in the system.
B43-B48	Six high-order address lines.

Appendix 5

General Magnetic Recording Techniques

CASSETTE RECORDING TECHNIQUES

TYPE	DATA RATE	ENCODING*
KANSAS CITY (SWTP)	300 BAUD	FSK 1 = 2400 Hz 0 = 1200 Hz
MITS	300 BAUD	FSK 1 = 2025 Hz 0 = 2225 Hz
TARBELL	1800 BAUD	PE 1 = 4800 Hz 0 = 2400 Hz
APPLE II	1500 BAUD	FSK 1 = 1000 Hz 0 = 2000 Hz
TRS-80 LEVEL 1	250 BAUD	
TRS-80 LEVEL II	500 BAUD	
PET	1100 BAUD	FSK

*FSK is frequency shift keying and PE is phase encoding.

References:

Jenkins, D., and McCauley, J. "Cassette Interfacing." *Kilobaud*, January 1979, pp. 78–81.

Tarnopol, L. "Care and Feeding of Cassette Tapes." *Kilobaud*, December 1978, pp. 44–46, and February 1979, pp. 106–107.

Wells, R. "PET's First Report Card." *Kilobaud*, May 1978, pp. 22–30.

TYPES OF FLEXIBLE (FLOPPY) DISKS

Size: 8-inch standard floppy
5¼-inch minifloppy

Sides: *Single-sided*
Data can be recorded on one side only.

Double-sided (reversible)
Data can be recorded on both sides and the diskette is removed and turned over to use the other side.

Density: Data is recorded at a specifiied number of bits per inch.

Single density for 8-inch disk, 77 track — 3408 bits per inch.

Double density for 8-inch disk, 77 track — 6816 bits per inch.

Note: Diskettes that are rated for double-density use are also usable at single density. The reverse is not generally true.

Sectoring: *Soft sector*
One hole is used to indicate the start of a track. The tracks are divided into sectors via software.

Hard sector
Each sector is indicated by a hole in the track. The number of holes is equal to the number of sectors per track. 8-inch disks are usually divided into 32 sectors, whereas 5¼-inch disks are usually divided into 10 or 16 sectors.

Appendix 6 Television Signal Standards

The Electronics Industry Association RS-170 standard establishes scan frequencies, signal voltage levels, and polarities for monochrome television as follows:

Number of lines per frame	525
Vertical scan frequency	60 Hz
Horizontal scan frequency	15,750 Hz
Black = negative voltage	
White = positive voltage	
Composite signal amplitude	1 V peak to peak into 75 ohms

All the relevant standards are available from

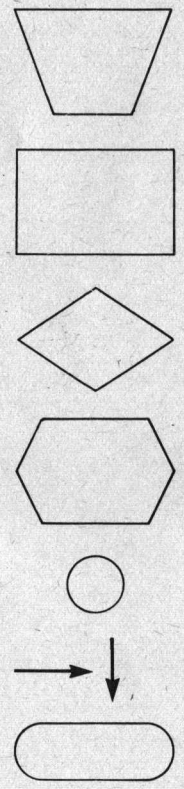
Engineering Department
Electronic Industries Association
2001 Eye St. N.W.
Washington, DC 20006

Among the important standards are:

RS-170	Monochrome Television Studio Facilities
RS-312-A	Engineering Specification Format for Monochrome CCTV Camera Equipment
RS-330	Electrical Performance Standards for Closed Circuit Television Camera 525/60 Interlaced 2:1
RS-343-A	Electrical Performance Standards for High Resolution Closed Circuit Television Cameras
RS-375-A	Electrical Performance Standards for Direct View Monochrome Closed Circuit TV Monitors 525/60 Interlaced 2:1

Appendix 7

Standard Flowchart Symbols



Input/Output

Processing—Arithmetic, logic, data movement

Decision logic

Subroutine

Connector point

Connector arrows

Terminal point (beginning or ending)

Index

A

Abstractions, in BASIC programs, 60, 81-84, 89
Accessing a data base, 37, 162
Accounting applications, 8, 44, 48, 102-106, 144, 149, 160, 196, 204-206, 228-229
  case study, 228-229
  payroll program example, 102-104
  printer required, 160
  tax calculation example, 105-106
Acoustic coupler, 163
Address, 14, 20-24, 62, 125
Addressable cursor, 141
Addressable reconstruction voice output method, 161
Advertisements, 208, 218, 219
AIM microcomputer (Rockwell), 31, 41, 180, 227
Aligning columns in output, 63, 67, 74
Alpha Miro (computer manufacturer), 34, 45, 204
Altos Computer Systems (manufacturer), 45, 51, 185, 225
Aluminized paper, 153
Analog input/output, 161
Apple Computer (manufacturer), 25, 28, 51, 180, 181, 256, 262, 265
  cassette recording technique, 265
  I/O bus, 256, 262
Appliance computers, 38, 42-45, 57-58, 180-181, 223, 224, 226, 227-228, 229
  additions, 45, 180-181
  advantages over calculators, 44
  advantages over single large computer, 228
  applications, 44-45, 224, 226, 227-228
  comparison with modular computers, 48
  cost, 42, 45
  expansion, 44-45, 180-181
  interfaces, 180-181, 182
  introductory system, 42-43
  software, 43-44
  tasks, 44-45, 224, 226-228
  uses, 224, 226
Applications of computers, 6-11, 29-30, 203-207, 223-229
Applications software, 35-36
Arithmetic operations, in BASIC, 61-62
Arithmetic quiz, program example, 59-60, 89
  averaging, 90-91
  dummy problem, 72
  grading, 78, 91-93
  interactive version, 77-78, 90-93
  pass/fail determination, 91-93
  random problems, 97
  repeated versions, 66-67, 68-69, 71-72, 88
  simple version, 62-63
Arrays, in BASIC programs, 89-97, 98, 113-114, 125
Art, 9-10
ASCII character code, 22, 137, 138, 248
  table, 248
Assembler, 36, 54-55
Assembly language, 35, 54-55, 172, 173, 205-206
Assignment statements, in BASIC, 61-62, 63, 65
Asterisk (multiplication sign in BASIC), 62, 65
Atari (computer manufacturer), 4, 27, 42, 51, 224
Attributes, on a CRT terminal, 142, 144
Audio cassette interface, 39, 42 (*see also* cassette interface)
Automatic transmission analogy (to a compiler), 56
Average (of a set of data), 90-91

B

Background of user, 208-209
Backup of media, 149, 150-151, 191-193, 195
BASIC (computer language), 55, 59-99, 120-121, 124, 125-129
  advantages, 55, 59
  commands, 61
  disadvantages, 55-56, 120
  features, 60-61
  functions, 97-98
  high-level language, 55-56
  interaction, 77-81, 89
  interpreter, 55-56
  looping, 71-72, 84-89
  names, 61-62, 64-65, 121
  operation signs, 62, 65
  order of operations, 61, 63, 64
  random numbers (RND function), 97-98
  statements, 61-62, 64-66, 68, 70, 76-77, 85-87, 110
  subroutines, 89, 90, 98, 110-112
  syntax checking, 124
  variables, 61, 65
Basic skills in ROM, 33
Beginner systems, 38, 42, 206-209, 215, 219, 227-228
Bill Godbout Electronics, 50
Billings Computer (Manufacturer), 45, 51
Bit, 28
Blank line, in a BASIC program, 63
Blink, on a CRT terminal, 142, 144
Bootstrap (loader), 33, 54
Branch (jump) instructions, 19, 124, 128-129
Branch point, in BASIC programs, 70
Breakpoint, 57, 122-123
Buffer memory, 167
Bug (error), 101
Building interfaces, 169-170
Building kits, 209, 211

Bundling, of software and hardware, 48, 58, 208, 228
Burning (programming), a (P)ROM, 53
Bus analyzer, 199
Buses, 14, 50-52, 170-171, 211, 252-264
  Apple I/O, 256, 262
  GPIB (IEEE-488), 170, 253-254
  Heath (Benton Harbor), 50, 256, 259
  IEEE-488 (GPIB), 170, 253-254
  KIM, 256, 263
  OSI-48, 50, 256, 265
  Radio Shack, 256, 260
  S-100, 50, 51, 52, 171, 211, 256, 257-258
  SS-50, 50, 256, 261
Business and financial applications, 8, 9, 10-11, 44, 48, 102-104, 105-106, 144, 149, 160, 196, 204-206, 226, 228-229
  case study, 228-229
  payroll program example, 102-104
  printer required, 160
  simulation, 10-11
  tax calculation example, 105-106
Buying a computer, 1, 203-217, 223-230
Byte, 28
Byte magazine, 218, 219
Bytesaver (Cromemco ROM board), 53
Byte Shop (chain of computer stores), 220

C

Calculator-like keyboards, 39, 140
Calculators, comparison with computers, 17, 18, 44
Camera interface, 162
Capacitors, 250
Card reader, 137
Carriage return character, 21, 22
Case, 32, 180-181
Cassette interface, 50, 137, 145-146, 265
Cassette operating system, 54, (*see also* operating system)
Cassette recorder, 145-146, 176, 190-191, 265
Cassettes, 34, 39, 42, 137, 145-146, 176, 190-191, 265
  advantages, 145-146
  disadvantages, 146
  erasing, 191
  interfaces, 176, 265
  length, 146, 190
  maintenance, 190-191
  serial access, 146
  software, 42
  use, 190-191
Categories of computers, 25
Categories of personal computers, 38, 57-58, 223-229
Central processing unit (CPU), 13, 14 (*see also* microprocessor)
Centronics interface, for printers, 156, 177, 254-255
Changing statement numbers, in BASIC programs, 61
Character codes, 20-25, 137-138, 248 (*see also* ASCII)
Character editor, 56
Character printer, 155
Checklist for debugging programs, 124-125
Chip, 12 (*see also* microprocessor)
Choosing a computer, 204-205, 208-209, 211-214, 216-217, 223-229
  case studies, 226-229
Circuit board, 39, 166
Classroom use of computers, 7-9, 42, 140, 227-228
Clearing the computer, in BASIC, 61
Clubs, 211, 212-213, 220, 222
Cluttered systems, 45, 180-181
COBOL (computer language), 55-56
Code, conversion, 166
  for characters, 137
Coding (writing computer instructions), 101, 120-121
Coding answers, 77, 81
Color, use of, 10, 144
Color graphics, 44, 143, 144
Color television, 44
Columns, alignment of, 63, 67
Commands, in BASIC, 61
Commas, in BASIC PRINT statements, 62, 67, 74
Comments, in programs, 70, 76, 132-133
Commodore PET computer, 1, 2, 42, 170, 176, 180, 181, 223, 265
Common programming errors, 124-125, 129
Common sense, in computer programs, 74, 76
Communications (telephone) interface, 5, 45, 162
Compatibility, 37-38, 52, 57-58, 211, 214
Compiler, 36, 55-56, 124
Complexity of computers, 123-124
Compucolor Corporation (computer manufacturer), 42, 43
Computer-aided-instruction, 7-8, 44, 205-206, 218, 228-229
Computer clubs, 211, 212-213, 220, 222
Computer errors, 74
Computer Faire (West Coast), 221
Computerland (chain of computer stores), 220-221
Computer language, 35
Computer magazines, 6, 217-220, 228
Computer shows, 221-222
Computer stores, 1, 220-222
Computers, appliance, 38, 42-45, 57-58, 180-181, 223-229
  applications of, 6-11, 29-30, 203-207, 223-229
  buying, 1, 203-217, 223-230
  categories of, 25
  categories of personal, 38, 57-58, 223-229
  classroom use of, 7-8, 9, 42, 140, 227-228
  costs of, 25, 207-208, 215-216
  differences among, 25-30
  environment for, 185-186
  errors by, 74
  history of, 11-13, 25
  keyboards for, 15, 39, 42, 137-140, 174, 193
  learning about, 204-205, 208, 215-216, 217-223
  maintenance of, 190-202
  mathematics and, 18
  modular, 45-50, 57-58, 181-182, 190, 206, 224, 226, 228-229

one-board, 38-41, 57-58, 140, 178-180, 182, 205, 206, 223, 224
operations of, 19-20
origin of, 18
places to buy, 1, 220-223
popularity of, 211-214
reliability of, 190
repair of, 196-200
sections of, 13-15
selecting, 203-230
setup of, 186-188
speed of, 25, 27-28
tasks for, 6-11
uses of, 6-11, 29-30
Computer's lack of insight and experience, 17
Computing Teacher magazine, 218-219, 228
Conditional form, in structured programming, 117-120
Conditions, in BASIC, 70, 76
Confusing letters and numbers, 121
Consecutive execution of instructions, 19
Consecutive form, in structured programming, 117-118
Control functions, of a peripheral, 165, 167-168
Control keys, on a keyboard, 139
Controller applications, 41, 51, 54, 204-206, 224, 226-227, 229
case study, 226-227
COSMAC VIP computer (RCA), 40, 41, 178
Costs, estimating, 207-208
of computers, 25, 207-208, 215-216
of memory, 52
of peripherals, 15, 29, 45, 152-153, 164
of software, 29, 42, 44-45, 49, 101
Counter, 68-69
Counting, in BASIC programs, 68-69, 76, 88
CP/M (Control Program/Microcomputer) operating system, 49, 51, 52, 54, 177, 200
CPU (Central Processing Unit), 13, 14, 32, 33, 50-52, 58
function, 13, 14
types, 50-52
Creative Computing magazine, 218-219
Cromemco (computer manufacturer), 2, 45-46, 49, 51, 53, 136, 172
CRT (cathode-ray-tube) terminal, 15, 141-144 (*see also* terminal, video display, video terminal)
Cultural bias, 25
Cursor, 141, 194

D

Daisy-wheel printer, 153, 156
Data arrays, in BASIC programs, 60, 89-97, 98
Data base, 37
Data base management system, 37
Data cassettes, 146, 190
Data flowchart, 106-107, 110, 132
DATA statement, in BASIC, 65-68
Data validation, 142
Debugger, 36, 57, 58, 122-130
Debugging (programs), 101, 122-130
approach, 129-130
breakpoints, 57, 122-123
checklist, 124-125
dump, 57, 122-123
example, 125-129
locating errors, 122-125
methods, 129-130
recommended approach, 129-130
single-step, 57, 122, 123
tools, 57, 122
trace, 57, 122-123, 126
Decisionmaking, in BASIC programs, 60, 70-76
Defensive programming, 120-121
Definition lists, 132
Deleting a line, in BASIC, 61
Design and development methods, for programs, 101, 107-108, 129-130, 131, 133-134
Diablo (printer manufacturer), 153
Diagnostics (test programs), 195, 197
Differences among computers, 25-30
peripherals, 29
price, 25, 27
software, 29
speed, 25-28
uses, 29-30
word length, 28-29
Digital cassettes, 146, 190
Digital control interface, 162
Digital Research (CA), 49 (*see also* CP/M operating system)
Digitizer, 162-163
DIM (dimension) statement, in BASIC, 90, 98, 113-114
Diodes, 250
Direct memory access (DMA), 177
Disk, 15-16, 34, 45, 54, 137, 146-152, 176-177, 192-193, 204
backup, 149, 150-151
choice of, 151-152
cost, 34, 147, 150
floppy, 15, 137, 146-149, 151-152
hard, 150-151
interfaces, 176-177
maintenance, 192-193
operating system, 54, 149, 150
purpose, 15-16
use, 192-193
Disk operating system (DOS), 54, 149, 150 (*see also* CP/M operating system, operating system)
Diskette, 147-149, 192-193, 266
care of, 192-193
cost, 147
storage capacity, 149
types, 147-148, 266
Display, calculator-like, 39
Division sign, in BASIC, 62, 65
DMA (direct memory access), 177
Documentation, manuals, 189-190, 197, 208
software, 101, 131-134
"Dog that didn't bark" logic, 76
Do-it-yourself, 197-202, 209, 211
kit building, 169-170, 209, 211
maintenance, 190-194
repair, 197-200

DOS (disk operating system), 54, 149, 150 (*see also* CP/M operating system)
Dot-matrix printer, 153-154, 158-159, 166
  method of forming characters, 158
Double-density floppy disk, 149, 266
Double disk computer systems, 149
Double-sided floppy disk, 149, 266
DO-WHILE statement, in structured programming, 118-120
Dr. Dobb's Journal magazine, 218-219
Driver (I/O program), 34, 36, 165-166, 171-173
Dummy problem, terminator, 72, 92, 95, 103
Dump, 57, 122-123
Duplicate names in programs, 64
Duplicating printer output, 154

E

Echoing characters, 140
Edge connector, 39
Editor (program), 36, 56-57, 61
Education, 7-8, 41, 44, 140, 227-228
80 Microcomputing magazine, 219
Electric Pencil word processing system (program) 50-52
Electrical noise, 185, 186
Electrosensitive paper, 153
Electrostatic printer, 153-154, 159, 160
Element numbers, in arrays, 90, 98
ELF computer (netronics), 41
END statement, in BASIC, 61-62
Ending marker, 72, 92, 95, 103
Engineering applications, 11, 38, 41, 44, 48, 204-229
  case study, 226-227
ENTER key, 61
Entering a BASIC program, 61, 63
Entering a name, 21-25
Entry-level computer system, 38, 39, 42, 44, 206-229
Environment, for a computer, 185-186
  for repair, 197-198
Erasing a line, in BASIC, 61
Error handling, by programs, 102-104
Error messages, 65, 68-69, 124
Errors, by computers, 74
Errors, locating in programs, 124-130
Estimating computer costs, 207-208
Evaluation kits, 41, 206, 227
Executing a BASIC program, 61
Exerciser, 56
Exidy Sorcerer computer, 42, 51, 138, 181
Expandibility, 45, 47, 180-181, 206, 211, 214
Extender board, 199, 201
External mass storage, 34 (*see also* cassette, disk, floppy disk)

F

Factors in choosing a computer, 31, 204-205, 208-209, 211-214, 216-217, 229
FALSE (same as 0 or OFF), 28
Fanfold paper, 153
File, 53, 57
Finkel, L., 222
Fixed-head disk, 150-152, 204 (*see also* hard disk)
Flexible disk, *see* floppy disk
Floppy disk, 15, 17, 45, 48, 146-149, 151-152, 177-178, 192-193, 266
  advantages, 147
  backup, 149, 192, 193
  capacity, 147-149
  characteristics, 17, 146-149
  choice of, 151-152
  cost, 45, 147
  disadvantages, 147
  DOS, 54, 149-150
  double-disk computer systems, 149
  interfaces, 177-178
  maintenance, 192-193
  media, 147-148, 177-178, 266
  operating system, 54, 149, 150
  picture, 17
  single-disk computer systems, 149
  storage of, 192
  storage capacity, 149
  types, 147, 149, 193, 266
Flowcharts, 105-109, 132
  advantages, 105
  data, 106-107, 110, 132
  documentation, 132
  examples, 105-106
  levels of detail, 132
  recommended use, 107-108
  set of symbols, 105, 107, 268
  symbols, 105, 107, 265
  use in documentation, 132
Forms handling, in BASIC programs, 63
  on printers, 153-154, 156, 160
  on video terminals, 142, 144
FOR-NEXT statement (in BASIC), 84-89
FORTRAN (computer language), 55, 120, 121
Friction feed, on printers, 153-154
Fully-formed characters, by printers, 153-154, 156, 160
Futre needs, 206 (*see also* expandibility)

G

Games, 6-7, 11, 204-205
Global Specialties (instrument manufacturer), 198-199
Godbout Electronics, 50
GOSUB statement, in BASIC, 110-111
GOTO statement, in BASIC, 66-68
GPIB (IEEE-488) interface, 253-254
Grading answers, 78, 80, 90-93
Graphics, 44, 141, 157
"Greater than" sign, in BASIC, 70
Green, W., 219
Guidelines, for choosing a computer, 204-205, 208-209, 211-214, 216-217, 229
  for documentation of programs, 133-134
  for environment, 186
  for maintaining programs, 202
  for modularizing programs, 116-117
  for programming projects, 101
  for testing programs, 130-131

H

Hand calculator analogy, 17-18
Hard copy, 142, 152-153
Hard disk, 15, 16, 150-152, 204
  backup, 150-151
  capacity, 150
  disadvantages, 150
  types, 150
Hard sectored floppy disk, 266
Hardware, 16, 32-34, 50-53
Hardware interface, 166-171, 173
Hardware/software tradeoffs, in interfacing, 173
Headings, 63, 67
Heath Corporation, bus, 50, 256, 259
  computers, 5, 36, 41, 189, 205, 207, 227
  home-study BASIC course, 99
  instruments, 198-200, 210
  printer, 154
  trainer, 41, 205, 227
Hexadecimal conversion table, 247
Hide (blank) feature, on a CRT terminal, 142
High-level language, 35, 54-56, 172, 173, 206
Highlighting, on a CRT terminal, 142, 144
History of computers, 11-13, 25
Hobby-grade instruments, 198-200
Hobby helper, 10
Home, of a cursor, 141
Home entertainment center, 6
Home-study courses, 99, 197
Housings, 45, 168
Human factors, 60, 77, 81, 89, 102-104, 140
Human qualities, in computer programs, 20-25, 60
Humidity, 185

I

IC (integrated circuit), 11-13, 250, 251
Identical systems, 195
IEEE-488 (GPIB) interface, 170, 253-254
IF-THEN statement, in BASIC, 70-76
IF-THEN-ELSE statement, in structured programming, 118-120
Impact printer, 153-154, 156, 159, 160
Improving the computer's environment, 186
Imsai (former computer manufacturer) 168, 184, 201
Incompatible commands, 38
Index, in arrays, 90-91
Information-hiding principle, 166
Information, sources of, 217-223
Info/World (computer newspaper), 218-219
Initial operation, of a computer, 188-190
INPUT statement, in BASIC, 77-81, 89
  comparison with READ statement, 79
Input/output (I/O) section, 13-15, 165-182
Inserting lines, in BASIC, 61
Inserting spaces, in BASIC printing, 63, 67
Installation of a computer, 186-188
Instructions, 19
Instruments, for computer repair, 197-200
Integrated circuits, 11-13, 250, 251
Integration of programs, 108-110
Intel Corporation, 12, 13
  8080 microprocessor, 12, 51
  8086 microprocessor, 51
Intelligence, in computer operations, 60, 70, 176
Intelligent peripheral, 169, 176-177, 182
Interactive languages, 55
Interactive programs, in BASIC, 77-81
Interconnections (buses), 14, 50
Interface Age magazine, 218-219
Interfaces, 32, 34, 156, 165-182
  appliance computer, 180-181, 182
  availability, 165, 167, 173, 182
  building of, 169-170
  cassette, 176, 265
  choice of, 174
  cost, 166, 167, 170, 180
  definition, 32, 34
  disk, 177-178
  floppy disk, 177-178
  functions, 165-166
  hardware, 166-171, 173
  hardware/software tradeoffs, 173
  I/O boards, 173
  keyboard, 174
  lack of, 156, 165, 182
  modular computer, 181-182
  need for, 165-166
  one-board computer, 178-179
  parallel, 156, 170, 171, 173
  printer, 156, 176-177
  reasons for complexity, 165-170
  serial, 170, 176
  software, 171-173
  standard, 170-171, 252-264
  terminal, 175-176
  testing, 170, 173
  video, 167, 175
Interpreter (translator program), 36, 56
Invalid answers, 76
Inventory program example, 59-60, 89
  interactive version, 78-80, 82
  negative levels, 74
  purchase orders with item numbers, 92, 94-96
  reorder point, 73-74
  repeated version, 67, 69, 88
  simple version, 63-64
Inverse video (on a CRT terminal), 142, 144
Inverting branches, 124, 128-129
I/O (input/output) boards, 173
I/O driver (program), 34, 36, 165-166, 171-173
I/O port, 14-15
I/O section, 13-15, 165-182 (*see also* interfaces)
I/O utility (program), 34, 36, 165-166, 171-173
Isolating operating problems, 195

J

Jump instructions, 19

K

K (unit of memory = 1024), 32
Kansas City standard (cassette interface), 176, 265

Katz, A., 222
Kernighan, B. W., 120
Keyboards, 15, 39, 42, 137-140, 174, 193
  advantages, 140
  calculator-like, 39, 140
  codes, 137-138
  criteria, 140
  disadvantages, 140
  features for ease of use, 138-140
  interfaces, 174
  maintenance, 193
  separation from printer, 140
  standards, 137
  types of, 137, 140
  use of, 193
Keypunch, 137
Kilobaud Microcomputing magazine, 219
Kilobyte (KB), 32
KIM one-board computer (MOS Technology), 40-41, 54, 178, 256, 263
  I/O bus, 256, 263
Kits, 169-170, 209, 211
KSR (keyboard send/receive) terminal, 157

L

Large-scale-integration (LSI), 13
Larsen, D., 222
Learning about computers, 204-205, 208, 215-216, 217-223
"Less than" sign, in BASIC, 70
LET statement, in BASIC, 61-62, 68
Letter-quality printer, 45, 153-154, 156, 160
Libes, S., 212, 222
Library, of programs and data, 116-117, 133-134, 145
Library forms, for programs, 133-134
Limit checking, on data, 74, 76, 111, 114-116
Line editor, 56
Line numbers, in BASIC programs, 60-61, 63-64, 66, 68, 70, 76
Line printer, 155
Lining up values and variables, 65, 66, 68
LIST command, in BASIC, 61
Literal execution of instructions, by computers, 123
Loading a program, 19, 146-147
Locating errors, in programs, 124-130
Logic analyzer, 199-200
Logic probe, 198-199
Long-term planning, 211, 214-216
Loop form, in structured programming, 117-120
Loop variable, in BASIC FOR-NEXT statements, 85-86
Loops, in programs, 84-89, 117-120
Low-cost prinbters, 158, 160, 194
Low-level language, 54 (*see also* assembly language)
LSI (large-scale integration), 13

M

Machine language, 35, 54 (*see also* assembly language, object code)
Magazines, 6, 217-220
Magnetic disk, 15 (*see also* disk, floppy disk, hard disk)
Magnetic tape unit, 137, 151
Mainframe (computer), 25-30, 183
Mail-order suppliers, 222-223
Maintenance (of computers), 190-202
  cassettes, 190-191
  contracts, 196-197, 200, 202, 228
  CRT, 193-194
  disks, 192-193
  floppy disks, 192-193
  isolating problems, 195
  keyboard, 193
  log, 195
  printer, 157, 160, 194
  recognizing problems, 194-195
  repair, 196-200
  service contracts, 196-197, 200, 202, 228
  software, 101, 200, 202
  video display, 193-194
Maintenance (of programs), 101-200, 202
Making decisions, in BASIC programs, 60, 70-76
Manuals, 188, 189-190, 197, 208
Mark-sense card reader, 117, 162
Mass distribution of software, 29
Mass storage, 34, 144-145 (*see also* cassette, disk)
Mathematics and computers, 18
Maximum value program, 125-129
Meaningful names, 64, 121
Mechanical arm analogy (to interfaces), 165
Media, 34, 137, 144-152, 190-193 (*see also* cassette, diskette)
Memory, 13-14, 32-33, 52-53, 173, 204 (*see also* RAM, ROM)
Memory map, 132-133
Memory sections, 13-14
Michael Shrayer Software, 50
Microcomputer, 13, 25
Microcomputing magazine, 219
Microprocessor, 13, 50-52, 227, 251
Microsoft (software company), 49
Mid-America Personal Computer Show (Chicago, IL), 221
Minicomputer, 25, 27-30
Mini-disk, 147-148
Minifloppy, 147-148
Modular computers, 45-50, 57-58, 181-182, 190, 206, 224, 226, 228-229
  accessories, 47-48
  applications, 48, 224, 226
  buses, 50
  characteristics, 45, 47
  comparison with appliance computers, 48
  cost, 45, 48, 229
  definition, 45, 57-58
  entry-level system, 45, 47
  expansion, 47-50, 182
  independent suppliers, 49-50
  interfaces, 181-182
  packaged versions, 48, 228-229
  software, 48-50
Modular programming, 108, 116-117
Module (part of a program), 108, 116-117
Monitor (program), 36, 54 (*see also* operating system)

Monopoly program example, 59, 60, 89
moving around the board, 74-76, 88-89
multiple players, 90, 96-97
property ownership, 81, 83-84
purchasability of properties, 84, 85
random moves, 98
repeated version, 67, 69, 88-89
simple version, 64
startup, 96
More memory, need for, 52
Morrow's Micro-Stuff, 50
MOS Technology KIM microcomputer, 40-41, 54, 178, 256, 263
MOS Technology 6502 microprocessor, 13, 51, 54
Mostek Corporation, 13
Motherboard, 47
Motorola, 13
MEK6800D2 microcomputer, 227
6800 microprocessor, 13, 33, 51
6809 microprocessor, 51
68000 microprocessor, 51
Multiplication sign, in BASIC, 62, 65
Multiterminal adapter, 163
Multi-user software, 163
Music, 9
Musical analogy (to computer programs), 16-17

N

Names, choice of, 121
National Computer Conference (NCC) Personal Computing Festival, 221
NEC Information Systems (printer manufacturer), 153, 156, 157, 160
Netronics ELF microcomputer, 41
NEW command, in BASIC, 61
NEXT statement, in BASIC, 84-89
NO and YES answers, in BASIC programs, 77, 81-82
Nonexecutable BASIC statements, 66, 70, 90 (*see also* DATA, DIM, REM statements)
Non-impact printer, 153, 156
Non-numerical concepts, in BASIC programs, 60, 81-84, 89
North Star (computer manufacturer), 51, 177-178, 224
"Not equal" sign, in BASIC, 70, 77
Numbering of lines, in BASIC, 60-61
Numeric pad, on a keyboard, 138-139, 140
Numerical addresses in memory, 14, 20-23, 62

O

Object code, 54-55
Off-line, 139
Ohio Scientific Instruments (computer manufacturer), 16, 17, 42, 45, 51, 256, 264
On-board memory, in peripheral interfaces, 167, 173
onComputing magazine, 218-219
On-line, 139
One-board computers, 38-41, 57-58, 140, 178-180, 182, 205-206, 223-224, 226-227, 229
additions, 39, 178-180, 206
applications, 41, 224, 226
case study, 226-227
components, 39
cost, 38, 41, 179, 180, 206
definition, 38, 57
interfaces, 178-180, 182
KIM, 40, 41, 54, 178, 256, 263
types, 39
uses, 224
Operating environment, 185-186
Operating procedures, for programs, 38, 102-104, 202
Operating system, 36, 53-54, 58, 124, 149, 151, 172
CP/M, 49, 51-52, 54, 177, 200
DOS, 54, 149-150
functions, 53-54
types, 54
Operation signs, in BASIC, 62, 65
Operations of a computer, 19-20
Order entry, 138, 139, 144
Order of execution, in BASIC programs, 60-61, 63-64, 66, 84
Order of variables, in BASIC DATA statements, 65, 66, 68
Origin of computers, 18
Oscilloscope, 170, 198, 200
OSI (Ohio Scientific Instruments) -48 bus, 50, 256, 264
OUT OF DATA error message, in BASIC, 65, 68-69
Overruns in software projects, 101
Packaged computers, 42, 45, 47-49, 181, 208, 216, 228-229
Page, time to print, 155
Paging, on a CRT terminal, 142
Paper, limiting use of, 152-153
special, 153, 154, 159, 160
Paper tape, 34, 161-162
reader, 161-162
Parallel interface, 156, 170-171, 173
Parallel port, 173
Parameter lists, 132
Paratronics (instrument manufacturer), 199-201, 210
Parroting a name, 21
PASCAL (computer language), 40, 55-56, 120-121
use with structured programming, 120
Payroll program example, 102-104, 105-106, 109
Peripherals, application requirements, 204-205
availability, 29
cost, 15, 29, 45, 152-153, 164
definition, 14
description, 136-164
driver programs, 34, 36, 165-166, 171-173
expansion, 45, 47-50, 168, 172, 178-182
interfaces, 165-182
maintenance, 190-194
standard devices with personal computers, 137

types, 14-15
use, 190-194
Personal Computing Show (East Coast), 221
Pertec (computer manufacturer), 45, 51, 150, 171, 207, 214-215, 226
PET computer (Commodore), 1, 2, 42, 170, 176, 180-181
Pickles and Trout (equipment manufacturer), 175
Places to buy a computer, 1, 220-221, 222-223
Plauger, P. J., 120
Playbook analogy (to computer programs), 35
Plotter, 162-163
Plotting on a printer, 157
Polymorphic Systems (computer manufacturer), 166
Popularity of computers, 211, 214
Port, 14-15, 173
Portability, of programs, 38, 56, 102
Precision required by a computer, 20, 22, 25, 63, 66, 123
Preprinted forms, 144, 153-154, 156
Price of computers, 1, 25, 207-208, 229
Primary features of BASIC language, 60-61
PRINT statement, in BASIC, 61-63
Printer, 45, 137, 152-161, 176-177, 194
changing character sets, 156
changing type face, 156
character, 155
character formation, 153, 158
character sets, 156
cost, 153, 158-159
daisy-wheel, 153, 156
dot-matrix, 153-154, 158-159, 166
electrostatic, 153-154, 159-160
extra-cost features, 158-159
formation of characters, 153, 158
friction feed, 153-154
fully-formed characters, 153-154, 156, 160
graphics, 157
impact, 149, 153-154, 160
interface, 156, 176-177
isolation, 156-157, 186
KSR model, 157
letter-quality, 45, 153-154, 156, 160
limiting use, 152-153
line, 155
low-cost, 158
maintenance, 157, 160-161, 194
multiple copies, 154
non-impact, 153, 156
paper size, 155
RO model, 157
Selectric, 154, 156, 160
speed, 154-155, 159, 160
teletypewriter, 154-157, 186
thermal, 31, 39, 153-154, 159-160
thimble, (Spinwriter), 153, 156-157, 160
tractor feed, 153-154
volume of output, 155-156, 194
width of paper, 155
Printing a message, 20-21, 23
Printing a name, 22-23
Printing terminal, 15, 157
Printing text, in BASIC, 62-63
Problem definition, 101, 102-104
Process of elimination, in program logic, 76
Processing requirements, of programs, 102-103
Processor, 13 (*see also* CPU, microprocessor)
Program design, 101, 104-120
aim, 105
design procedure, 108
flowcharting, 105-109
methods, 104-105
modular programming, 108, 116-117
objectives of methods, 105
structured programming, 108, 117-120
top-down design, 108-116
Program stubs, 109, 111, 112
Programmers, skills of, 18, 101
Programming a (P)ROM, 53
Programming errors, 124-125, 129
Programming procedures in BASIC, 60-61
Programming time, use of, 101
Programs, 16-17 (*see also* software)
locating errors in, 124-130
portability of, 38, 56, 102
processing requirements of, 102-103
relocation of, 132, 133
straightforward order in, 121
PROM programmer, 53
Protected fields (on a CRT terminal), 142, 144
Punched cards, 1, 137
Purchasing computers, case studies of, 226-229
Purchasing software, 16, 42, 44-45, 48-49, 52, 58, 100-101, 200, 202, 211, 216, 220

Q

Query (a data base), 37
Question mark, with BASIC INPUT statements, 77
Qume (printer manufacturer), 153, 155-156
Quotation marks, in BASIC, 62-64

R

Radio Shack, bus, 256, 260
instruments, 198
TRS-80 microcomputer, 1, 3, 42, 44, 51, 145, 162, 169, 176, 180-181, 187-188, 219, 256, 260
RAM (read/write memory), 32, 52, 58, 173, 204
Random access, to information on a disk, 15, 147
Random numbers, in BASIC programs, 97-98
use in testing programs, 130-131
RCA COSMAC VIP computer, 40-41, 178
READ statement, in BASIC, 65-68
comparison with INPUT statement, 79
Read-only memory (ROM), 32-33, 53, 58, 173
Read/write memory (RAM), 32, 52, 58, 173
Real-time applications, 27-28
Real-time clock, 162
Recognizing a character, 22, 24-25
Recognizing problems, in computer operations, 194-195
Relational operators, in BASIC, 70, 76-77
Reliability, of computers, 190

Relocation, of programs, 132-133
REM (remark) statement, 70, 76
Remembering a name, 21-22, 23-24
Remote data bases, 37, 45
Repair, of computers, 196-200
Repair environment, 197-200
Repeating statements, in BASIC programs, 66-68
RESET switch, 20
Resistor color code, 249
Resistors, 249
Restoring operations, of computers, 194-195, 197
RETURN key, 61
RETURN statement, in BASIC, 110, 112
Reverse video, on a CRT terminal, 142
Ridiculous results, 74
RND function, in BASIC, 97-98
RO (receive-only) terminal, 157
Rockwell AIM-65 microcomputer, 31, 41, 180, 227
Rollover, 139
ROM (read-only memory), 32-33, 53, 58, 173
Rony, P., 222
RS-232 interface, 170, 252-253
RUN command, in BASIC, 61, 63
Running out of data, in BASIC programs, 66, 68, 72

S

Sales tactics, 45, 216
Savings files, 195
Schools, 222
SCR (scratch) command, in BASIC, 61
Screen editor, 56
Scrolling, on a CRT, 50, 141
Search, 57
Sections of a computer, 13-15
Sectoring, of floppy disks, 266
Selecting a computer, 203-230
  applications, 203-207
  background, 208-209
  case studies, 226-229
  cost, 207-208
  criteria, 203-206, 211-217, 224-229
  do-it-yourself, 209-211
  expansion, 206, 211, 214-216
  factors, 203-206, 211-217
  goals, 211-217
  mail-order suppliers, 222-223
  occupation, 224-229
  price, 207-208
  reliability, 205
  sources of information, 217-223
  stores, 220-221
  time to buy, 215-216
Selectric printers, 154, 156, 160
Self-test programs, 195, 197
Semicolons, in BASIC PRINT statements, 62, 67, 74
Semiconductor technology, 11-13
Sequential execution of BASIC programs, 60-61, 63, 66, 68, 73-74, 84
Serial access (of cassettes), 146
Serial interface, 170, 176 (*see also* RS-232 interface)
Serial printer, 155
Service call, 196, 197
Service contract, 194, 196-197, 200, 202
Sets of numbers, in BASIC programs, 89-98, 113-114, 125
Setup, of a computer, 186-188
Short cassettes, 146, 190
Shows (computer), 221-222
Shrayer Software, 50
Simulation, 10-11, 27
Single-board computers, 38-41, 57-58, 140, 178-180, 182, 205-206, 223-224, 226-227, 229 (*see also* one-board computers)
Single-density floppy disk, 148-149, 266
Single-disk computer systems, 149
Single-step, 57, 122-123
Single-user systems, 140
Skipping lines on a printer or CRT, in BASIC, 63
Skipping statements, in BASIC programs, 66, 68, 73-74, 84, 125
Slot (inside a computer), 45-46, 168, 182
Smart peripherals, 169
Softech Microsystems (suppliers of UCSD PASCAL), 50
Soft-sectored floppy disk, 266
Software, 16-17, 29, 35-36, 38, 48-135, 149, 165, 171-173, 200, 202, 205-206, 228-229
  additions, 44-45, 48-50
  applications, 35-36
  assemblers, 36, 54-55
  availability, 29
  BASIC programming, 59-99
  compatibility, 38
  compilers, 36, 55-56
  cost, 29, 42, 44-45, 48-49
  debuggers, 36, 57
  design methods, 101, 104-120
  development, 100-135
  editors, 36, 56-57
  interface, 165, 171-173
  interpreters, 36, 55-56
  maintenance, 101, 200, 202
  operating system, 36, 53-54, 149
  package, 48-49, 228
  projects, 101
  purpose, 16-17
  systems, 35-36, 53-58, 102, 122-124, 172
S-100 bus, 50-52, 171, 211, 256-258
S-100 Microsystems magazine, 218, 256
Source code, 54-55
Sources of information, 217-223
Southwest Technical Products Corporation (computer manufacturer), 1, 3, 46, 47, 50-51, 139, 225, 256, 261
Spaces, inserting of, in BASIC programs, 63, 67
Spacing of line numbers, in BASIC programs, 60-61
Special paper, for printers, 153, 154, 159-160
Speech synthesizer, 5, 161-162
Speed of computers, 25, 27-28
Spinwriter (NEC printer), 153, 156-157, 160
SS-50 bus, 50, 256, 261
Standard floppy disk, 147-148, 266
Standard interfaces, 170-171, 252-264 (*see also* RS-232, S-100 bus)

Standards, 38, 50, 101, 105, 107, 137, 146, 170-172, 211, 252-264
Starting address, from RESET, 20
Statements, in BASIC, 60, 64
Static charge, 190
Status, of a peripheral, 165, 167-168
Status line, on a video terminal, 142
Stone tablet analogy (to ROM), 32, 53
String variables, 81
Straightforward order, in programs, 121
Structured programming, 108, 117-120
  advantages, 120
  forms of, 117-120
Stubs, 109, 111-112
Subroutine call, 110
Subroutines, in BASIC programs, 110, 111-116
Suitability for a task, 31-32, 203-206, 223-229
Summation program, 85-87
Symbols, in BASIC programs, 63-64
Synertek SYM-1 microcomputer, 41, 223
Syntax checking, 124, 129
Sysgen (system generation), 53
System commands, in BASIC, 61
System generation, 53
Systems software, 35, 53-58, 102, 122-124, 172

T

Tabs, 63
Take-home computer, 38
Tape unit, 137, 151
Tarbell cassette interface, 176-177, 265
Tarbell Electronics, 50, 176-177, 179
Teacher, 8, 9, 227-229
  case study, 227-228
Teaching, 7-8, 41, 44, 206, 218-219, 224, 226-228
Telephone interface, 5, 45, 162
Teletypewriter, 154-157, 186
Television set, 1, 141, 175, 265
Television signal standards, 267
Terminal, 15, 45, 141-144, 157, 167, 175, 193-194
  cost, 45
  definition, 15
  home-brew, 175-176
  interfaces, 167, 175
  maintenance, 193-194
Terminator, 72, 92, 95, 103
Test scoring program example, 111-116
Testing programs, 101, 130-131
Texas Instruments (computer manufacturer), 5, 42
THEN clause, in BASIC, 70, 76
Theraml printer, 31, 39, 153-154, 159-160
Thimble printer (Spinwriter), 153, 156-157, 160
Thinker Toys (equipment manufacturer), 50, 148, 174
Time to buy, 215-216
Time to print a page, 155
Titus, J., 222
Tools for computer repair, 197-198, 211
Top-down design, 108-116
Trace, 57, 122, 123, 126, 128, 130
Tractor feed, on printers, 153-154
Trainer, 41, 44, 226-227
Transistors, 250
Trenton (NJ) Computer Festival, 222
Trivial cases, 124-125, 130
TRS-80 microcomputer (Radio Shack), 1, 3, 42, 44, 51, 145, 162, 169, 176, 180-181, 187-188, 219, 256, 260
TRUE (same as 1 or ON), 28
Type ball, 150, 156
Typed character, 22, 24
Typewriter, 140, 153, 157, 160, 167
Typing a message, 20-23
Typing skill, 140

U, V

UCSD PASCAL, 40
Unit of memory (K), 32
Unreasonable results, 74, 76
Unused variable, 124, 126
Updating a variable, in BASIC, 64, 68
Upgrading older systems, 211, 214
User interaction, in BASIC programs, 60, 77-81
Uses of computers, 6-11, 29-30
Utility program (*see* I/O utility)
Valid response, 77, 81
Variable names, in BASIC, 61, 64, 65
Vector Graphic (computer manufacturer), 32, 52, 138, 143, 175, 187
Version numbers, of popular programs, 200, 202
Video converter, 141
Video digitizer, 162
Video display, 137, 141-144
  advantages, 142
  choice, 143-144
  color versions, 144
  cost, 143
  disadvantages, 142
  features, 141-142
  interface, 167
  maintenance, 193-194
Video display boards, 143, 167, 175
Video interface, 167, 175
Video (RAM) memory, 167
Video modulator, 141
Video monitor, 141, 175
Video terminal, 15, 153
Voice input/output devices, 5-6, 161-163
Volatile memory, 32
Voltmeter, 198-199
Volume setting, on cassette recorders, 146, 190

W, Y, Z

Warranty service, 196, 216
Warren, J., 218
West Coast Computer Faire, 221
Word length, 28-29
Word processing, 8-9, 36, 139, 153, 160, 205
Word processing system, 9, 36 (*see also* Electric Pencil word processing system)
Workstation, 186
Write protect tab, on a cassette or diskette, 191-192
YES and NO answers, in BASIC programs, 77, 81-82
Zilog Corporation, 13
  Z-80 microprocessor, 13, 51
  Z-8000 microprocessor, 51

The most reliable, in-depth guide to purchasing, using, programming, & maintaining your computer... $8.95

WHY DO YOU NEED A PERSONAL COMPUTER?

Lance A. Leventhal & Irvin Stafford

Whether you're considering buying a personal computer or already own one, this non-technical, down-to-earth introduction to the world of personal computers will help you get the most from your machine for home, recreational, educational, or business use.

Why Do You Need A Personal Computer? is both an invaluable consumer handbook and a practical owner's manual. Packed with savvy tips, concrete examples, and specific know-how, it shows you what kind of personal computer is best suited to your needs ... helps you operate and maintain your computer ... and enables you to take full advantage of its hundreds of amazing practical applications. No other guide on the market today is as comprehensive, practical, and user-oriented as this one.

Features include:

■ Sensible, detailed guidelines on the advantages and drawbacks of every type of personal computer ■ Sources of equipment and information ■ A step-by-step introduction to the BASIC language, together with an easy-to-follow course in writing programs ■ Information on how to maintain your computer, and what to do if it breaks down ■ A user-oriented discussion of peripherals and interfacing ■ An extensive glossary ■ And more

Find out how remarkably powerful and versatile personal computers are. Learn the many everyday applications personal computers can perform, what you will need for a useful system, and how to select a system to meet your requirements. Low-cost personal computers are now widely advertised as invaluable for educational, business, financial, scientific, and hobby applications. **This book will help you sort out the numerous offerings and understand what the advertisements and sales pitches really mean.**

If you have been intrigued by the idea of owning your own computer or have been wondering whether a personal computer might be a worthwhile investment, this book will explain the possibilities, the alternatives, and the problems. You don't have to be a mathematician or an engineer to read this book and learn to take advantage of this new, powerful technology. **Open it up, and begin!**

Lance A. Leventhal, Ph.D., is an independent consultant specializing in microprocessors and small computers and the author of 10 books and over 60 articles on those subjects.

Irvin Stafford is a Senior Engineer with Burroughs Corporation, Advanced Systems Development Organization. He has 8 years experience in the design, construction, and use of personal computers and microprocessors.

Look for these and other popular microcomputer paperbacks from Wiley at your favorite bookstore or computer shop:

Albrecht, Finkel, & Brown—**BASIC, 2nd ed.**
Albrecht, Finkel, & Brown—**BASIC for Home Computers**
Albrecht, Inman, & Zamora—**TRS-80 BASIC**
Albrecht, Finkel, & Brown—**ATARI BASIC**
Friedmann, Greenberg, & Hoffberg—**FORTRAN IV, 2nd ed.**
Ashley—**ANS COBOL, 2nd ed.**
Ashley—**Structured COBOL**
Fernandez & Ashley—**Using CP/M**
Ashley & Fernandez—**Job Control Language**
Finkel & Brown—**Data Files in BASIC**
Harris—**Introduction to Data Processing, 2nd ed.**
Ashley—**Background Math for a Computer World, 2nd ed.**
Stern—**Flowcharting**
Leventhal & Stafford—**Why Do You Need a Personal Computer?**
McGlynn—**Personal Computing**
Ashley & Fernandez—**Introduction to 8080/8085 Assembly Language Programming**
Miller—**8080/Z80 Assembly Language: Techniques for Improved Programming**
Inman, Zamora, & Albrecht—**TRS-80 Intermediate to Advanced BASIC**

JOHN WILEY & SONS, INC.
605 Third Avenue, New York, N.Y. 10158
New York • Chichester • Brisbane • Toronto

ISBN 0 471 04784-8

Cover Illustration: Jerry McDaniel